# W3School MongoDB 教程

wizardforcel

Published with GitBook



# 目錄

# W3School MangoDB教程

来源：MongoDB教程

整理：飞龙

# NoSQL 简介

NoSQL(NoSQL = Not Only SQL )，意即"不仅仅是SQL"。

在现代的计算系统上每天网络上都会产生庞大的数据量。

这些数据有很大一部分是由关系数据库管理系统（RDMBSs）来处理。 1970年 E.F.Codd's提出的关系模型的论文 "A relational model of data for large shared data banks"，这使得数据建模和应用程序编程更加简单。

通过应用实践证明，关系模型是非常适合于客户服务器编程，远远超出预期的利益，今天它是结构化数据存储在网络和商务应用的主导技术。

NoSQL 是一项全新的数据库革命性运动，早期就有人提出，发展至2009年趋势越发高涨。NoSQL的拥护者们提倡运用非关系型的数据存储，相对于铺天盖地的关系型数据库运用，这一概念无疑是一种全新的思维的注入。

## 关系型数据库遵循ACID规则

事务在英文中是transaction，和现实世界中的交易很类似，它有如下四个特性：

**1、A (Atomicity)** 原子性
原子性很容易理解，也就是说事务里的所有操作要么全部做完，要么都不做，事务成功的条件是事务里的所有操作都成功，只要有一个操作失败，整个事务就失败，需要回滚。

比如银行转账，从A账户转100元至B账户，分为两个步骤：1）从A账户取100元；2）存入100元至B账户。这两步要么一起完成，要么一起不完成，如果只完成第一步，第二步失败，钱会莫名其妙少了100元。

**2、C (Consistency)** 一致性
一致性也比较容易理解，也就是说数据库要一直处于一致的状态，事务的运行不会改变数据库原本的一致性约束。

例如现有完整性约束a+b=10，如果一个事务改变了a，那么必须得改变b，使得事务结束后依然满足a+b=10，否则事务失败。

**3、I (Isolation)** 独立性
所谓的独立性是指并发的事务之间不会互相影响，如果一个事务要访问的数据正在被另外一个事务修改，只要另外一个事务未提交，它所访问的数据就不受未提交事务的影响。
比如现有有个交易是从A账户转100元至B账户，在这个交易还未完成的情况下，如果此时B查询自己的账户，是看不到新增加的100元的。

**4、D (Durability)** 持久性
持久性是指一旦事务提交后，它所做的修改将会永久的保存在数据库上，即使出现宕机也不会丢失。

# 分布式系统

分布式系统（distributed system）由多台计算机和通信的软件组件通过计算机网络连接（本地网络或广域网）组成。

分布式系统是建立在网络之上的软件系统。正是因为软件的特性，所以分布式系统具有高度的内聚性和透明性。

因此，网络和分布式系统之间的区别更多的在于高层软件（特别是操作系统），而不是硬件。

分布式系统可以应用在在不同的平台上如：Pc、工作站、局域网和广域网上等。

# 分布式计算的优点

可靠性（容错）：
分布式计算系统中的一个重要的优点是可靠性。一台服务器的系统崩溃并不影响到其余的服务器。

可扩展性：
在分布式计算系统可以根据需要增加更多的机器。

资源共享：
共享数据是必不可少的应用，如银行，预订系统。

灵活性：
由于该系统是非常灵活的，它很容易安装，实施和调试新的服务。

更快的速度：
分布式计算系统可以有多台计算机的计算能力，使得它比其他系统有更快的处理速度。

开放系统：
由于它是开放的系统，本地或者远程都可以访问到该服务。

更高的性能：
相较于集中式计算机网络集群可以提供更高的性能（及更好的性价比）。

# 分布式计算的缺点

故障排除： ：
故障排除和诊断问题。

软件：
更少的软件支持是分布式计算系统的主要缺点。

网络：
网络基础设施的问题，包括：传输问题，高负载，信息丢失等。

安全性：
开发系统的特性让分布式计算系统存在着数据的安全性和共享的风险等问题。

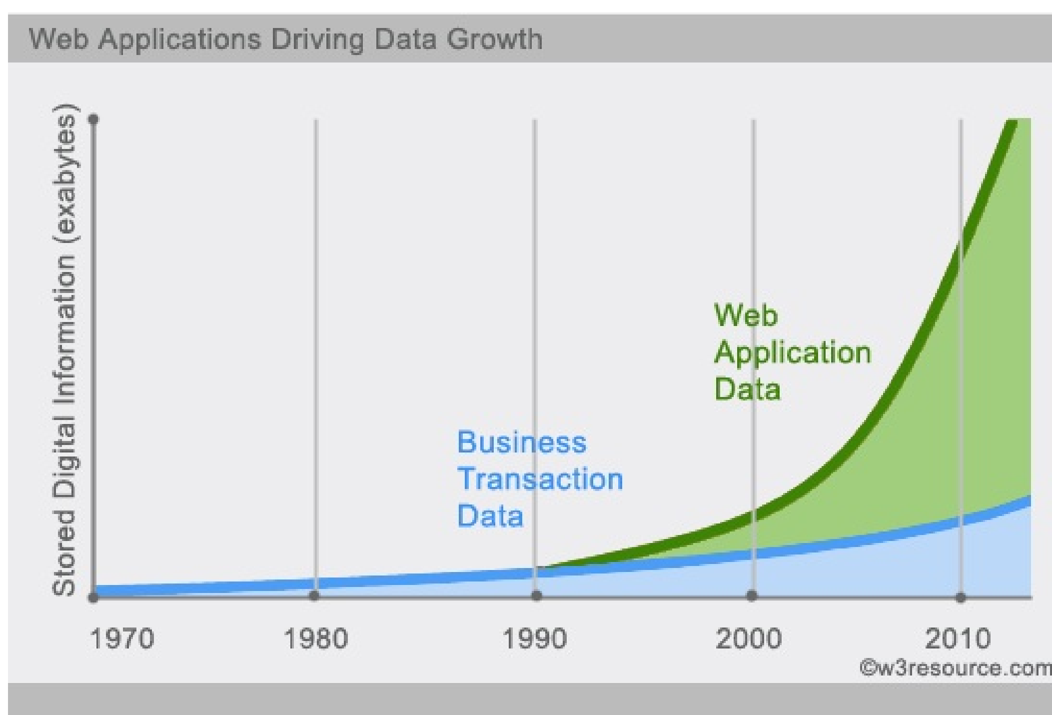
## 什么是NoSQL?

NoSQL，指的是非关系型的数据库。NoSQL有时也称作Not Only SQL的缩写，是对不同于传统的关系型数据库的数据库管理系统的统称。

NoSQL用于超大规模数据的存储。（例如谷歌或Facebook每天为他们的用户收集万亿比特的数据）。这些类型的数据存储不需要固定的模式，无需多余操作就可以横向扩展。

## 为什么使用NoSQL ?

今天我们可以通过第三方平台（如：Google,Facebook等）可以很容易的访问和抓取数据。用户的个人信息，社交网络，地理位置，用户生成的数据和用户操作日志已经成倍的增加。我们如果要对这些用户数据进行挖掘，那SQL数据库已经不适合这些应用了, NoSQL数据库的发展也却能很好的处理这些大的数据。



# 实例

社会化关系网:

```
Each record: UserID1, UserID2
Separate records: UserID, first_name,last_name, age, gender,...
Task: Find all friends of friends of friends of ... friends of a given user.
```

Wikipedia 页面 :

```
Large collection of documents
Combination of structured and unstructured data
Task: Retrieve all pages regarding athletics of Summer Olympic before 1950.
```

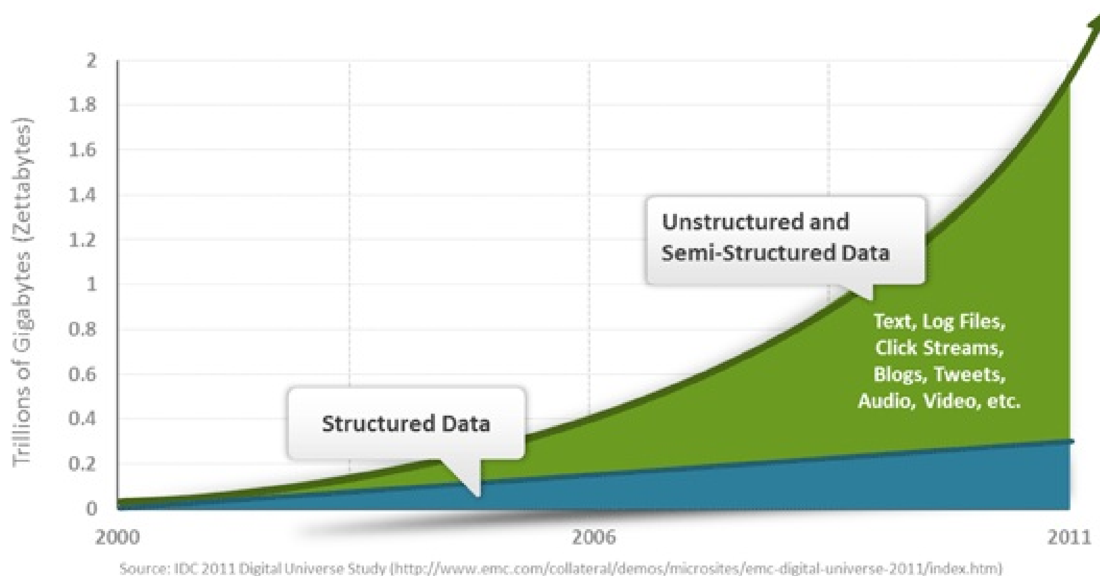
# RDBMS vs NoSQL

### RDBMS

- 高度组织化结构化数据
- 结构化查询语言（SQL） (SQL)
- 数据和关系都存储在单独的表中。
- 数据操纵语言，数据定义语言
- 严格的一致性
- 基础事务

### NoSQL

- 代表着不仅仅是SQL
- 没有声明性查询语言
- 没有预定义的模式
  -键 - 值对存储，列存储，文档存储，图形数据库
- 最终一致性，而非ACID属性
- 非结构化和不可预知的数据
- CAP定理
- 高性能，高可用性和可伸缩性




# NoSQL 简史

NoSQL一词最早出现于1998年，是Carlo Strozzi开发的一个轻量、开源、不提供SQL功能的关系数据库。

2009年，Last.fm的Johan Oskarsson发起了一次关于分布式开源数据库的讨论[2]，来自Rackspace的Eric Evans再次提出了NoSQL的概念，这时的NoSQL主要指非关系型、分布式、不提供ACID的数据库设计模式。

2009年在亚特兰大举行的"no:sql(east)"讨论会是一个里程碑，其口号是"select fun, profit from real_world where relational=false;"。因此，对NoSQL最普遍的解释是"非关联型的"，强调Key-Value Stores和文档数据库的优点，而不是单纯的反对RDBMS。

# CAP定理（CAP theorem）

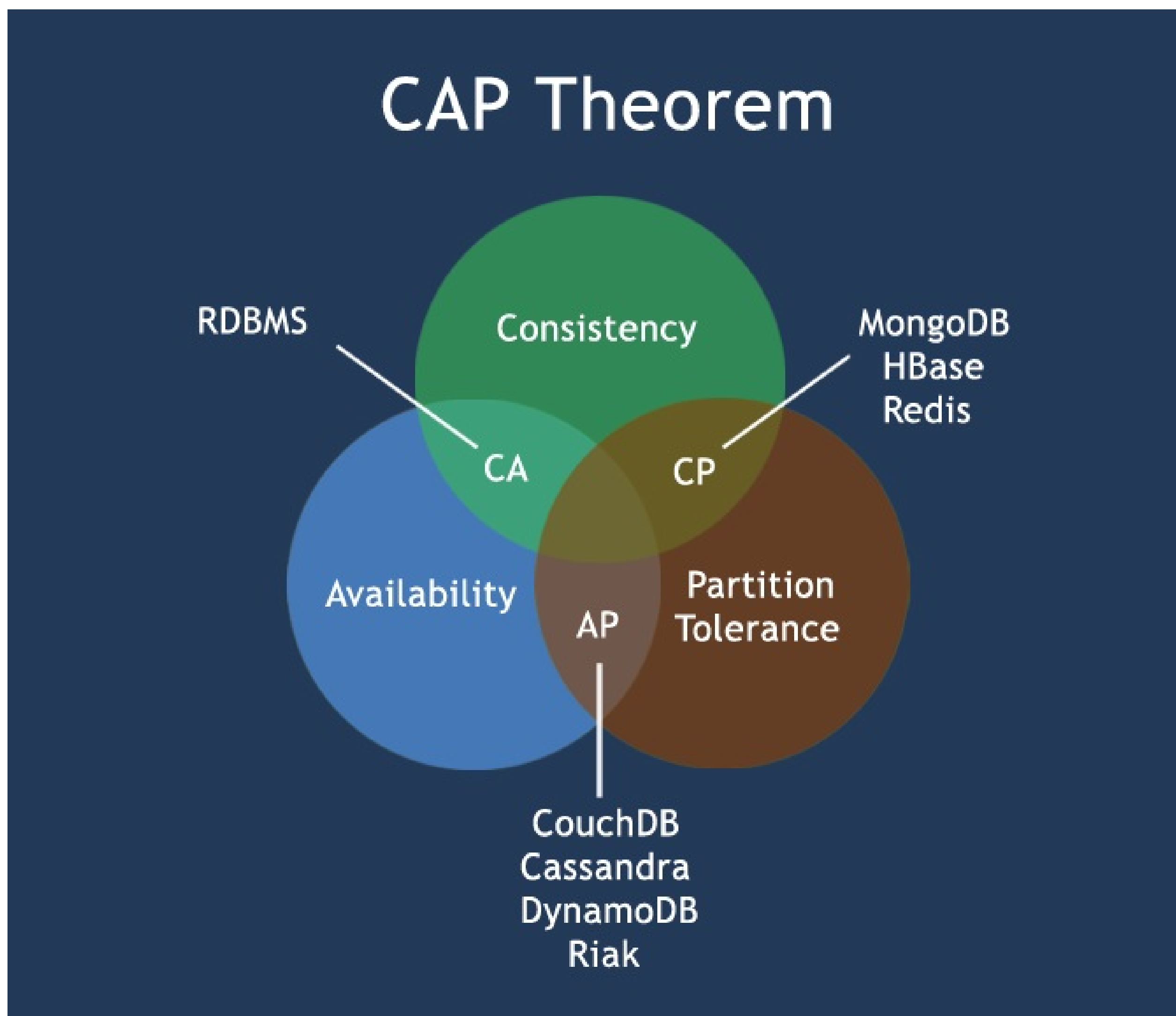
在计算机科学中, CAP定理（CAP theorem）, 又被称作 布鲁尔定理（Brewer's theorem）, 它指出对于一个分布式计算系统来说，不可能同时满足以下三点:

- 一致性(**Consistency**) (所有节点在同一时间具有相同的数据)
- 可用性(**Availability**) (保证每个请求不管成功或者失败都有响应)
- 分隔容忍(**Partition tolerance**) (系统中任意信息的丢失或失败不会影响系统的继续运作)

CAP理论的核心是：一个分布式系统不可能同时很好的满足一致性，可用性和分区容错性这三个需求，最多只能同时较好的满足两个。

因此，根据 CAP 原理将 NoSQL 数据库分成了满足 CA 原则、满足 CP 原则和满足 AP 原则三 大类：

- CA - 单点集群，满足一致性，可用性的系统，通常在可扩展性上不太强大。
- CP - 满足一致性，分区容忍必的系统，通常性能不是特别高。
- AP - 满足可用性，分区容忍性的系统，通常可能对一致性要求低一些。




## NoSQL的优点/缺点

优点:

- 
    - 高可扩展性
- 
    - 分布式计算
- 
    - 低成本
- 
    - 架构的灵活性，半结构化数据
- 

    - 没有复杂的关系

缺点:

-
    - 没有标准化
-
    - 有限的查询功能（到目前为止）
-
    - 最终一致是不直观的程序

# BASE

BASE：Basically Available, Soft-state, Eventually Consistent。 由 Eric Brewer 定义。

CAP理论的核心是：一个分布式系统不可能同时很好的满足一致性，可用性和分区容错性这三个需求，最多只能同时较好的满足两个。

BASE是NoSQL数据库通常对可用性及一致性的弱要求原则:

- Basically Availble --基本可用
- Soft-state --软状态/柔性事务。 "Soft state" 可以理解为"无连接"的, 而 "Hard state" 是"面向连接"的
- Eventual Consistency --最终一致性 最终一致性， 也是是 ACID 的最终目的。

# ACID vs BASE

| ACID | BASE |
|---|---|
| 原子性(**A**tomicity) | 基本可用(**B**asically **A**vailable) |
| 一致性(**C**onsistency) | 软状态/柔性事务(**S**oft state) |
| 隔离性(**I**solation) | 最终一致性 (**E**ventual consistency) |
| 持久性 (**D**urable) | |

# NoSQL 数据库分类

| 类型 | 部分代表 | 特点 |
| --- | --- | --- |
| 列存储 | Hbase Cassandra Hypertable | 顾名思义，是按列存储数据的。最大的特点是方便存储结构化和半结构化数据，方便做数据压缩，对针对某一列或者某几列的查询有非常大的IO优势。 |
| 文档存储 | MongoDB CouchDB | 文档存储一般用类似json的格式存储，存储的内容是文档型的。这样也就有有机会对某些字段建立索引，实现关系数据库的某些功能。 |
| key-value存储 | Tokyo Cabinet / Tyrant Berkeley DB MemcacheDB Redis | 可以通过key快速查询到其value。一般来说，存储不管value的格式，照单全收。（Redis包含了其他功能） |
| 图存储 | Neo4J FlockDB | 图形关系的最佳存储。使用传统关系数据库来解决的话性能低下，而且设计使用不方便。 |
| 对象存储 | db4o Versant | 通过类似面向对象语言的语法操作数据库，通过对象的方式存取数据。 |
| xml数据库 | Berkeley DB XML BaseX | 高效的存储XML数据，并支持XML的内部查询语法，比如XQuery,Xpath。 |

# 谁在使用

现在已经有很多公司使用了NoSQ：

- Google
- Facebook
- Mozilla
- Adobe
- Foursquare
- LinkedIn
- Digg
- McGraw-Hill Education
- Vermont Public Radio

# 什么是MongoDB ?

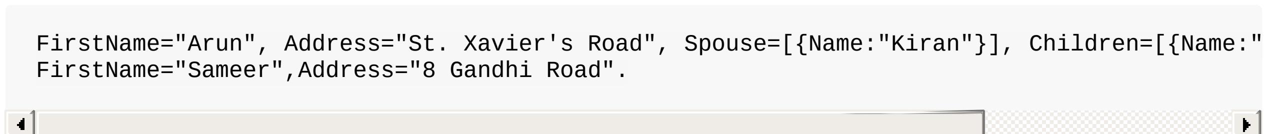
MongoDB 是由C++语言编写的开源数据库系统。

在高负载的情况下，添加更多的节点，可以保证服务器性能。

MongoDB 旨在为WEB应用提供可扩展的高性能数据存储解决方案。




MongoDB 将数据存储为一个文档。MongoDB是一个基于分布式文件存储的数据库。



```
FirstName="Arun", Address="St. Xavier's Road", Spouse=[{Name:"Kiran"}], Children=[{Name:"
FirstName="Sameer",Address="8 Gandhi Road".
```

注意：以上数据有两个不同的文档（以"."分隔）。以这种方式存储数据即为文件存储的数据库。 MongoDB是一个面向文档的数据库。

## 主要特点

- MongoDB的提供了一个面向文档存储，操作起来比较简单和容易。
- 你可以在MongoDB记录中设置任何属性的索引 (如：FirstName="Sameer",Address="8 Gandhi Road")来实现更快的排序。
- 你可以通过本地或者网络创建数据镜像，这使得MongoDB有更强的扩展性。
- 如果负载的增加（需要更多的存储空间和更强的处理能力），它可以分布在计算机网络中的其他节点上这就是所谓的分片。
- Mongo支持丰富的查询表达式。查询指令使用JSON形式的标记，可轻易查询文档中内嵌的对象及数组。
- MongoDb 使用update()命令可以实现替换完成的文档（数据）或者一些指定的数据字段。
- Mongodb中的Map/reduce主要是用来对数据进行批量处理和聚合操作。
- Map和Reduce。Map函数调用emit(key,value)遍历集合中所有的记录，将key与value传给Reduce函数进行处理。
- Map函数和Reduce函数是使用Javascript编写的，并可以通过db.runCommand或mapreduce命令来执行MapReduce操作。
- GridFS是MongoDB中的一个内置功能，可以用于存放大量小文件。

- MongoDB允许在服务端执行脚本，可以用Javascript编写某个函数，直接在服务端执行，也可以把函数的定义存储在服务端，下次直接调用即可。
- MongoDB支持各种编程语言:RUBY，PYTHON，JAVA，C++，PHP，C#等多种语言。
- MongoDB安装简单。

历史 2007年10月，MongoDB由10gen团队所发展。2009年2月首度推出。

- 2012年05月23日，MongoDB2.1 开发分支发布了! 该版本采用全新架构，包含诸多增强。
- 2012年06月06日，MongoDB 2.0.6 发布，分布式文档数据库。
- 2013年04月23日，MongoDB 2.4.3 发布，此版本包括了一些性能优化，功能增强以及bug修复。
- 2013年08月20日，MongoDB 2.4.6 发布，是目前最新的稳定版。

MongoDB 下载

你可以在mongodb官网下载该安装包，地址为：http://www.mongodb.org/downloads。MonggoDB支持以下平台:

- OS X 32-bit
- OS X 64-bit
- Linux 32-bit
- Linux 64-bit
- Windows 32-bit
- Windows 64-bit
- Solaris i86pc
- Solaris 64

# MongoDB 工具

有几种可用于MongoDB的管理工具。

## 监控

MongoDB提供了网络和系统监控工具Munin，它作为一个插件应用于MongoDB中。

Gangila是MongoDB高性能的系统监视的工具，它作为一个插件应用于MongoDB中。

基于图形界面的开源工具 Cacti, 用于查看CPU负载, 网络带宽利用率,它也提供了一个应用于监控 MongoDB 的插件。

## GUI

- Fang of Mongo – 网页式,由Django和jQuery所构成。
- Futon4Mongo – 一个CouchDB Futon web的mongodb山寨版。
- Mongo3 – Ruby写成。
- MongoHub – 适用于OSX的应用程序。
- Opricot – 一个基于浏览器的MongoDB控制台, 由PHP撰写而成。
- Database Master — Windows的mongodb管理工具
- RockMongo — 最好的PHP语言的MongoDB管理工具，轻量级, 支持多国语言.

# MongoDB 应用案例

下面列举一些公司MongoDB的实际应用：

- Craiglist上使用MongoDB的存档数十亿条记录。
- FourSquare，基于位置的社交网站，在Amazon EC2的服务器上使用MongoDB分享数据。
- Shutterfly，以互联网为基础的社会和个人出版服务，使用MongoDB的各种持久性数据存储的要求。
- bit.ly, 一个基于Web的网址缩短服务，使用MongoDB的存储自己的数据。
- spike.com，一个MTV网络的联营公司， spike.com使用MongoDB的。
- Intuit公司，一个为小企业和个人的软件和服务提供商，为小型企业使用MongoDB的跟踪用户的数据。
- sourceforge.net，资源网站查找，创建和发布开源软件免费，使用MongoDB的后端存储。
- etsy.com ，一个购买和出售手工制作物品网站，使用MongoDB。
- 纽约时报，领先的在线新闻门户网站之一，使用MongoDB。
- CERN，著名的粒子物理研究所，欧洲核子研究中心大型强子对撞机的数据使用MongoDB。

# window平台安装 MongoDB

## MongoDB 下载

MongoDB提供了可用于32位和64位系统的预编译二进制包，你可以从MongoDB官网下载安装，MongoDB预编译二进制包下载地址：

http://www.mongodb.org/downloads

| | OS X 32-bit note | OS X 64-bit | Linux 32-bit note | Linux 64-bit | Windows 32-bit note | Windows 64-bit | Solaris i86pc note | Solaris 64 | Source |
|---|---|---|---|---|---|---|---|---|---|
| Production Release (Recommended) | | | | | | | | | |
| 1.8.2 6/17/2011 Changelog Release Notes | download | download | download *legacy-static | download *legacy-static | download | download | download | download | tgz zip |

## 解压

下载zip包后，解压安装包，并安装它。

创建数据目录

MongoDB将数据目录存储在 db 目录下。但是这个数据目录不会主动创建，我们在安装完成后需要创建它。请注意，数据目录应该抽奖在根目录下（(如： C:\ 或者 D:\ 等 )。

在本教程中，我们已经在D：盘中解压了mongodb文件，现在让我们创建一个data的目录然后在data目录里创建db目录。

```
D:\>mkdir data
D:\>cd data
D:\data>mkdir db
D:\data>cd db
D:\data\db>
```

你也可以通过window的资源管理器中创建这些目录，而不一定通过命令行。

## 命令行下运行 MongoDB 服务器

为了从命令提示符下运行MongoDB服务器，你必须从MongoDB目录的bin目录中执行mongod.exe文件。

```
D:\mongodb>cd bin

D:\mongodb\bin>mongod
mongod --help for help and startup options
Thu Jul 07 12:59:48 [initandlisten] MongoDB starting : pid=2448 port=27017 dbpat
h=/data/db/ 32-bit

** NOTE: when using MongoDB 32 bit, you are limited to about 2 gigabytes of data

**       see http://blog.mongodb.org/post/137788967/32-bit-limitations
**       with --dur, the limit is lower

Thu Jul 07 12:59:48 [initandlisten] db version v1.8.1, pdfile version 4.5
Thu Jul 07 12:59:48 [initandlisten] git version: a429cd4f535b2499cc4130b06ff7c26
f41c00f04
Thu Jul 07 12:59:48 [initandlisten] build sys info: windows (5, 1, 2600, 2, 'Ser
vice Pack 3') BOOST_LIB_VERSION=1_35
Thu Jul 07 12:59:49 [initandlisten] waiting for connections on port 27017
Thu Jul 07 12:59:49 [websvr] web admin interface listening on port 28017
```

# 将MongoDB服务器作为Windows服务运行

请注意，你必须有管理权限才能运行下面的命令。执行以下命令将MongoDB服务器作为Windows服务运行：

mongod --bind_ip yourIPadress --logpath "C:\data\dbConf\mongodb.log" --logappend --dbpath "C:\data\db" --port yourPortNumber --serviceName "YourServiceName" --serviceDisplayName "YourServiceName" --install

下表为**mongodb**启动的参数说明：

| 参数 | 描述 |
| --- | --- |
| --bind_ip | 绑定服务IP，若绑定127.0.0.1，则只能本机访问，不指定默认本地所有IP |
| --logpath | 定MongoDB日志文件，注意是指定文件不是目录 |
| --logappend | 使用追加的方式写日志 |
| --dbpath | 指定数据库路径 |
| --port | 指定服务端口号，默认端口27017 |
| --serviceName | 指定服务名称 |
| --serviceDisplayNam | 指定服务名称，有多个mongodb服务时执行。 |
| --install | 指定作为一个Windows服务安装。 |

# MongoDB后台管理 Shell

如果你需要进入MongoDB后台管理，你需要先打开mongodb装目录的下的bin目录，然后执行mongo.exe文件，MongoDB Shell是MongoDB自带的交互式Javascript shell,用来对MongoDB进行操作和管理的交互式环境。

当你进入mongoDB后台后，它默认会链接到 test 文档（数据库）：

```
D:\>cd mongodb

D:\mongodb>cd bin

D:\mongodb\bin>mongo
MongoDB shell version: 1.8.1
connecting to: test
>
```

由于它是一个JavaScript shell，您可以运行一些简单的算术运算:

```
D:\>cd nongodb

D:\mongodb>cd bin

D:\mongodb\bin>mongo
MongoDB shell version: 1.8.1
connecting to: test
> 2+2
4
>
```

db 命令先了当前操作的文档（数据库）：

```
D:\>cd mongodb

D:\mongodb>cd bin

D:\mongodb\bin>mongo
MongoDB shell version: 1.8.1
connecting to: test
> 2+2
4
> db
test
>
```

插入一些简单的记录并查找它：

```
D:\mongodb\bin>mongo
MongoDB shell version: 1.8.1
connecting to: test
> db.w3r.insert({x:10})
> db.w3r.find()
{ "_id" : ObjectId("4e156fdee1aa39b8e1e77fad"), "x" : 10 }
>
```

第一个命令将10插入到w3r集合的x字段中

# Linux平台安装MongoDB

## 下载

MongoDB提供了linux平台上32位和64位的安装包，你可以在官网下载安装包。

下载地址：http://www.mongodb.org/downloads

| | OS X 32-bit note | OS X 64-bit | Linux 32-bit note | Linux 64-bit | Windows 32-bit note | Windows 64-bit | Solaris i86pc note | Solaris 64 | Source |
|---|---|---|---|---|---|---|---|---|---|
| Production Release (Recommended) | | | | | | | | | |
| 1.8.2 6/17/2011 Changelog Release Notes | download | download | download *legacy-static | download *legacy-static | download | download | download | download | tgz zip |

## 安装

下载完成后，在你安装的目录下解压zip包。

## 创建数据库目录

MongoDB的数据存储在data目录的db目录下，但是这个目录在安装过程不会自动创建，所以你需要手动创建data目录，并在data目录中创建db目录。

注意：请将data目录创建于根目录下(/)。

File Edit View Terminal Tabs Help

```
debian:/# mkdir data
debian:/# cd data
debian:/data# mkdir db
debian:/data# cd db
debian:/data/db#
```

## 命令行中运行 MongoDB 服务

你可以再命令行中执行mongo安装目录中的bin目录执行mongod命令来启动mongdb服务。

```
debian:/home/ritwik# ./mongodb/bin/mongod
./mongodb/bin/mongod --help for help and startup options
Tue Jul 12 07:59:40 [initandlisten] MongoDB starting : pid=3715 port=27017 dbpath=/data/db/ 32-bit

** NOTE: when using MongoDB 32 bit, you are limited to about 2 gigabytes of data
**       see http://blog.mongodb.org/post/137788967/32-bit-limitations
**       with --dur, the limit is lower

Tue Jul 12 07:59:40 [initandlisten] db version v1.8.2, pdfile version 4.5
Tue Jul 12 07:59:40 [initandlisten] git version: 433bbaa14aaba6860da15bd4de8edf600f56501b
Tue Jul 12 07:59:40 [initandlisten] build sys info: Linux bs-linux32.10gen.cc 2.6.21.7-2.fc8xen #1 SMP Fri Feb
ST_LIB_VERSION=1_37
Tue Jul 12 07:59:40 [initandlisten] waiting for connections on port 27017
Tue Jul 12 07:59:40 [websvr] web admin interface listening on port 28017
```

# MongoDB后台管理 Shell

如果你需要进入MongoDB后台管理，你需要先打开mongodb装目录的下的bin目录，然后执行mongo命令文件。

MongoDB Shell是MongoDB自带的交互式Javascript shell,用来对MongoDB进行操作和管理的交互式环境。

当你进入mongoDB后台后，它默认会链接到 test 文档（数据库）：

```
debian:/home/ritwik# ./mongodb/bin/mongod
./mongodb/bin/mongod --help for help and startup options
Tue Jul 12 07:59:40 [initandlisten] MongoDB starting : pid=3715 port=27017 dbpath=/data/db/ 32-bit

** NOTE: when using MongoDB 32 bit, you are limited to about 2 gigabytes of data
**       see http://blog.mongodb.org/post/137788967/32-bit-limitations
**       with --dur, the limit is lower

Tue Jul 12 07:59:40 [initandlisten] db version v1.8.2, pdfile version 4.5
Tue Jul 12 07:59:40 [initandlisten] git version: 433bbaa14aaba6860da15bd4de8edf600f56501b
Tue Jul 12 07:59:40 [initandlisten] build sys info: Linux bs-linux32.10gen.cc 2.6.21.7-2.fc8xen #1 SMP Fri Feb
ST_LIB_VERSION=1_37
Tue Jul 12 07:59:40 [initandlisten] waiting for connections on port 27017
Tue Jul 12 07:59:40 [websvr] web admin interface listening on port 28017
```

由于它是一个JavaScript shell，您可以运行一些简单的算术运算:

File Edit View Terminal Tabs Help

```
debian:/home/ritwik# ./mongodb/bin/mongo
MongoDB shell version: 1.8.2
connecting to: test
> 3+5
8
> db
test
>
```

现在让我们插入一些简单的数据，并对插入的数据进行检索：

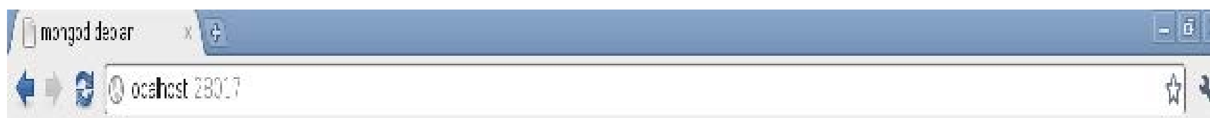
```
debian:/home/ritwik# ./mongodb/bin/mongo
MongoDB shell version: 1.8.2
connecting to: test
> 3+5
8
> db
test
> db.w3r.insert({z:8})
> db.w3r.find()
{ "_id" : ObjectId("4e1bb52ccb73fc0d2cf72818"), "z" : 8 }
>
```

第一个命令是将数据 8 插入到w3r集合（表）的 z 字段中。

# MongoDb web 用户界面

在比MongoDB服务的端口多1000的端口上，你可以访问到MondoDB的web用户界面。

如：如果你的MongoDB运行端口使用默认的27017，你可以在端口号为28017访问web用户界面。




# mongod debian

List all commands | Replica set status

Commands: buildinfo cursorInfo features isMaster listDatabases replSetGetStatus serverStatus top

db version v1.8.2, pdfile version 4.5
git hash: 433bbaa14aaba6860da15bd4de8edf600f56501b
sys info: Linux bs-linux32.10gen.cc 2.6.21.7-2.fc8xen #1 SMP Fri Feb 15 12:39:36 EST 2008 i686 BOOST_LIB_VERSION=1_37
uptime: 298 seconds

**low level** requires read lock

time to get readlock: 20ms
# databases: 2

replication:
master: 0
slave: 0
initialSyncCompleted: 1

**clients**

| Client | OpId | Active | LockType | Waiting | SecsRunning | Op | Namespace | Query | client | msg | progress |
|---|---|---|---|---|---|---|---|---|---|---|---|
| snapshotthread | 0 | | 0 | | | 0 | | | (NONE) | | |
| clientcursormon | 0 | | R | | | 0 | | | (NONE) | | |
| websvr | 0 | | 0 | | | 0 | | | (NONE) | | |
| conn | 9 | | 0 | | | 2004 | test.$cmd | { replSetGetStatus: 1, forShell: 1 } | 127.0.0.1:48593 | | |
| initandlisten | 0 | | W | | | 2004 | local.system.namespaces | { name: /^local.temp./ } | 0.0.0.0 | | |

**dbtop** (occurences/percent of elapsed)

| NS | total | | Reads | | Writes | | Queries | | GetMores | | Inserts | | Updates | | Removes | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| TOTAL | 3 | 58.2% | 2 | 0.4% | 1 | 55.8% | 2 | 0.4% | 0 | 0% | 1 | 55.8% | 0 | 0% | 0 | 0% |
| local.system.namespaces | 1 | 0.0% | 1 | 0.0% | 0 | 0% | 1 | 0.0% | 0 | 0% | 0 | 0% | 0 | 0% | 0 | 0% |

```
VM13721:1 Resource interpreted as Image but transferred with MIME type text/html: "http:/
```



# MongoDB 数据库，对象，集合

## 描述

不管我们学习什么数据库都应该学习其中的基础概念，在mongodb中基本的概念是文档、集合、数据库，下面我们挨个介绍。

## 数据库

一个mongodb中可以建立多个数据库。

MongoDB的默认数据库为"db"，该数据库存储在data目录中。

在MongoDB中可以创建数据库，如果你想使用MongoDB，创建数据库不是必要的。

"show dbs" 命令可以显示所有数据的列表。

```
D:\mongodb\bin>mongo
MongoDB shell version: 1.8.1
connecting to: test
> show dbs
admin   (empty)
comedy  0.03125GB
local   (empty)
student 0.03125GB
test    0.03125GB
> _
```

执行 "db" 命令可以显示当前数据库对象或者集合。

```
D:\mongodb\bin>mongo
MongoDB shell version: 1.8.1
connecting to: test
> db
test
> _
```

运行"use"命令，可以连接到一个指定的数据库。

```
> db
test
> use student
switched to db student
>
```

以上实例命令中，"student" 是你要检索的数据库。

在下一个章节我们将详细讲解MongoDB中命令的使用。

数据库名称可以是任何字符，但是不能包含空字符串，点号（.），或者" "。

"system" 作为系统保留字符串不能作为数据库名。

数据库名不能包含 "$"。

# 文档

文档是mongodb中的最核心的概念，是其核心单元，我们可以将文档类比成关系型数据库中的每一行数据。

多个键及其关联的值有序的放置在一起就是文档。在mongodb中使用一种类json的bson存储数据。

bson数据可以理解为在json的基础上添加了一些json中没有的数据类型。

如果我们会json，那么bson我们就已经掌握了一半了，至于新添加的数据类型后面我会介绍。

文档例子如下：

```
{ site : "w3cschool.cc" }
```

通常，"object（对象）" 术语是指一个文件。

文件类似于一个RDBMS的记录。

我们可以对集合（collection）进行插入，更新和删除操作。

下表将帮助您更容易理解Mongo中的一些概念：

| RDBMS | MongoDB |
| --- | --- |
| Table（表） | Collection（集合） |
| Column（栏） | Key（键） |
| Value（值） | Value（值） |
| Records / Rows（记录/列） | Document / Object（文档/对象） |

下表为MongoDB中常用的几种数据类型。

| 数据类型 | 描述 |
|---|---|
| string（字符串） | 可以是一个空字符串或者字符组合。 |
| integer（整型） | 整数。 |
| boolean（布尔型） | 逻辑值 True 或者 False。 |
| double | 双精度浮点型 |
| null | 不是0，也不是空。 |
| array | 数组：一系列值 |
| object | 对象型，程序中被使用的实体。可以是一个值，变量，函数，或者数据结构。 |
| timestamp | timestamp存储为64为的值，只运行一个mongod时可以确保是唯一的。前32位保存的是UTC时间，单位是秒，后32为是在这一秒内的计数值，从0开始，每新建一个MongoTimestamp对象就加一。 |
| Internationalized Strings | UTF-8 字符串。 |
| Object IDs | 在mongodb中的文档需要使用唯一的关键字_id来标识他们。几乎每一个mongodb文档都使用_id字段作为第一个属性（在系统集合和定容量集合（capped collection）中有一些例外）。_id值可以是任何类型，最常见的做法是使用ObjectId类型。 |

# 集合

集合就是一组文档的组合。如果将文档类比成数据库中的行，那么集合就可以类比成数据库的表。

在mongodb中的集合是无模式的，也就是说集合中存储的文档的结构可以是不同的，比如下面的两个文档可以同时存入到一个集合中：

```
{"name":"mengxiangyue"} {"Name":"mengxiangyue","sex":"nan"}
```

当第一个文档插入时，集合就会被创建。

# 合法的集合名

集合名称必须以字母或下划线开头。

集合名可以保护数字

集合名称不能使美元符"$"，"$"是系统保留字符。

集合的名字 最大不能超过128个字符 。

另外，"."号的使用在集合当中是允许的，它们被成为子集合(Subcollection)；比如你有一个blog集合，你可以使用blog.title，blog.content或者blog.author来帮组你更好地组织集合。

如下实例：

```
db.tutorials.php.findOne()
```

## capped collections

Capped collections 就是固定大小的collection。

它有很高的性能以及队列过期的特性(过期按照插入的顺序). 有点和 "RRD" 概念类似。

Capped collections是高性能自动的维护对象的插入顺序。它非常适合类似记录日志的功能 和标准的collection不同，你必须要显式的创建一个capped collection， 指定一个collection的大小，单位是字节。collection的数据存储空间值提前分配的。

要注意的是指定的存储大小包含了数据库的头信息。> db.createCollection("mycoll", {capped:true, size:100000})

- 在capped collection中，你能添加新的对象。
- 能进行更新，然而，对象不会增加存储空间。如果增加，更新就会失败 。
- 数据库不允许进行删除。使用drop()方法删除collection所有的行。
- 注意: 删除之后，你必须显式的重新创建这个collection。
- 在32bit机器中，capped collection最大存储为1e9( 1X109)个字节。

## 元数据

数据库的信息是存储在集合中。它们使用了系统的命名空间：

```
dbname.system.*
```

在MongoDB数据库中名字空间 <dbname>.system.* 是包含多种系统信息的特殊集合(Collection)，如下:

| 集合命名空间 | 描述 |
| --- | --- |
| dbname.system.namespaces | 列出所有名字空间。 |
| dbname.system.indexes | 列出所有索引。 |
| dbname.system.profile | 包含数据库概要(profile)信息。 |
| dbname.system.users | 列出所有可访问数据库的用户。 |
| dbname.local.sources | 包含复制对端（slave）的服务器信息和状态。 |

对于修改系统集合中的对象有如下限制。

在插入数据，可以创建索引。但除此之外该表信息是不可变的(特殊的drop index命令将自动更新相关信息)。

是可修改的。 是可删除的。

# MongoDB - 连接

## 描述

在本教程我们将讨论MongoDB的不同连接方式。

## 启动 MongoDB服务

在前面的教程中，我们已经讨论了如何启动MongoDB服务，你只需要在MongoDB安装目录的bin目录下执行'mongod'即可。

执行启动操作后，mongodb在输出一些必要信息后不会输出任何信息，之后就等待连接的建立，当连接被建立后，就会开始打印日志信息。

你可以使用MongoDB shell 来连接 MongoDB 服务器。你也可以使用PHP来连接mongodb。本教程我们会使用 MongoDB shell来连接Mongodb服务，之后的章节我们将会介绍如何通过php 来连接MongoDB服务。

默认情况下，MongoDB的启动端口为27017。比MongoDB启动端口大1000的端口为MongoDB的web用户界面，你可以再浏览器中输入 http://localhost:28017 来访问MongoDB的web用户界面。

## 通过shell连接MongoDB服务

你可以通过执行以下命令来连接MongoDB的服务。

注意：localhost为主机名，这个选项是必须的：

```
mongodb://localhost
```

当你执行以上命令时，你可以看到以下输出结果：

```
D:\mongodb\bin>mongo
MongoDB shell version: 1.8.1
connecting to: test
> mongodb://localhost
...
```

如果你检查从哪里连接到MongoDB的服务器，您可以看到如下信息：

```
D:\mongodb\bin>mongod
mongod --help for help and startup options
Wed Jul 13 14:14:51 [initandlisten] MongoDB starting : pid=2820 port=27017 dbpat
h=/data/db/ 32-bit

** NOTE: when using MongoDB 32 bit, you are limited to about 2 gigabytes of data

**       see http://blog.mongodb.org/post/137788967/32-bit-limitations
**       with --dur, the limit is lower

Wed Jul 13 14:14:51 [initandlisten] db version v1.8.1, pdfile version 4.5
Wed Jul 13 14:14:51 [initandlisten] git version: a429cd4f535b2499cc4130b06ff7c26
f41c00f04
Wed Jul 13 14:14:51 [initandlisten] build sys info: windows (5, 1, 2600, 2, 'Ser
vice Pack 3') BOOST_LIB_VERSION=1_35
Wed Jul 13 14:14:51 [initandlisten] waiting for connections on port 27017
Wed Jul 13 14:14:51 [websvr] web admin interface listening on port 28017
Wed Jul 13 14:15:52 [initandlisten] connection accepted from 127.0.0.1:53272 #1
```

最后一行（标记处），打印了你成功连接上MongoDB服务的信息。

## MongoDB连接命令格式

使用用户名和密码连接到MongoDB服务器，你必须使用 'username:password@hostname/dbname' 格式，'username'为用户名，'password' 为密码。

使用用户名和密码连接登陆到默认数据库：

```
mongodb://mongo_admin:AxB6_w3r@localhost/
```

以上命令中，用户 mongo_admin使用密码AxB6_w3r连接到本地的MongoDB服务上。输出结果如下所示：

```
D:\mongodb\bin>mongo
MongoDB shell version: 1.8.1
connecting to: test
> mongodb://mongo_admin:AxB6_w3r@localhost/
...
```

使用用户名和密码连接登陆到指定数据库：

连接到指定数据库的格式如下：

```
mongodb://mongo_admin:AxB6_w3r@localhost/w3r
```

## 更多连接实例

连接本地数据库服务器，端口是默认的。

```
mongodb://localhost
```

使用用户名fred，密码foobar登录localhost的admin数据库。

```
mongodb://fred:foobar@localhost
```

使用用户名fred，密码foobar登录localhost的baz数据库。

```
mongodb://fred:foobar@localhost/baz
```

连接 replica pair, 服务器1为example1.com服务器2为example2。

```
mongodb://example1.com:27017,example2.com:27017
```

连接 replica set 三台服务器 (端口 27017, 27018, 和27019):

```
mongodb://localhost,localhost:27018,localhost:27019
```

连接 replica set 三台服务器, 写入操作应用在主服务器 并且分布查询到从服务器。

```
mongodb://host1,host2,host3/?slaveOk=true
```

直接连接第一个服务器，无论是replica set一部分或者主服务器或者从服务器。

```
mongodb://host1,host2,host3/?connect=direct;slaveOk=true
```

当你的连接服务器有优先级，还需要列出所有服务器，你可以使用上述连接方式。

安全模式连接到localhost:

```
mongodb://localhost/?safe=true
```

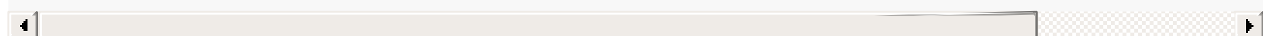
以安全模式连接到replica set，并且等待至少两个复制服务器成功写入，超时时间设置为2秒。

```
mongodb://host1,host2,host3/?safe=true;w=2;wtimeoutMS=2000
```

# 参数选项说明

标准格式：

```
mongodb://[username:password@]host1[:port1][,host2[:port2],...[,hostN[:portN]]][/[databas
```



标准的连接格式包含了多个选项(options)，如下所示：

| 选项 | 描述 |
| --- | --- |
| replicaSet=name | 验证replica set的名称。 Impliesconnect=replicaSet. |
| slaveOk=true\|false | true:在connect=direct模式下，驱动会连接第一台机器，即使这台服务器不是主。在connect=replicaSet模式下，驱动会发送所有的写请求到主并且把读取操作分布在其他从服务器。 false: 在 connect=direct模式下，驱动会自动找寻主服务器. 在connect=replicaSet 模式下，驱动仅仅连接主服务器，并且所有的读写命令都连接到主服务器。 |
| safe=true\|ffalse | true: 在执行更新操作之后，驱动都会发送getLastError命令来确保更新成功。(还要参考 wtimeoutMS). false: 在每次更新之后，驱动不会发送getLastError来确保更新成功。 |
| w=n | 驱动添加 { w : n } 到getLastError命令. 应用于safe=true。 |
| wtimeoutMS=ms | 驱动添加 { wtimeout : ms } 到 getlasterror 命令. 应用于 safe=true. |
| fsync=true\|ffalse | true: 驱动添加 { fsync : true } 到 getlasterror 命令.应用于 safe=true. false: 驱动不会添加到getLastError命令中。 |
| journal=true\|ffalse | 如果设置wie true, 同步到 journal (在提交到数据库前写入到实体中). 应用于 safe=true |
| connectTimeoutMS=ms | 可以打开连接的时间。 |
| socketTimeoutMS=ms | 发送和接受sockets的时间。 |

# PHP安装MongoDB扩展驱动

## 描述

本教程将向大家介绍如何在Linux、window、Mac平台上安装MongoDB扩展。

## Linux上安装 MongoDB PHP扩展

### 在终端上安装

你可以在linux中执行以下命令来安装MongoDB 的 PHP 扩展驱动

```
$ sudo pecl install mongo
```

使用php的pecl安装命令必须保证网络连接可用以及root权限。

安装手册

如果你想通过源码来编译扩展驱动。你必须手动编译源码包，这样做的好是最新修正的bug包含在源码包中。

你可以在Github上下载MongoDB PHP驱动包。访问github网站然后搜索"mongo php driver"(下载地址：https://github.com/mongodb/mongo-php-driver)，下载该源码包，然后执行以下命令：

```
$ tar zxvf mongodb-mongodb-php-driver-<commit_id>.tar.gz
$ cd mongodb-mongodb-php-driver-<commit_id>
$ phpize
$ ./configure
$ sudo make install
```

如果你的php是自己编译的，则安装方法如下(假设是编译在/usr/local/php目录中)：

```
$ tar zxvf mongodb-mongodb-php-driver-<commit_id>.tar.gz
$ cd mongodb-mongodb-php-driver-<commit_id>
$ /usr/local/php/bin/phpize
$ ./configure --with-php-config=/usr/local/php/bin/php-config
$ sudo make install
```

执行以上命令后，你需要修改php.ini文件，在php.ini文件中添加mongo配置，配置如下：

```
extension=mongo.so
```

注意：你需要指明 extension_dir 配置项的路径。

## window上安装 MongoDB PHP扩展

Github上已经提供了用于window平台的预编译php mongodb驱动二进制包(下载地址：https://s3.amazonaws.com/drivers.mongodb.org/php/index.html)，你可以下载与你php对应的版本，但是你需要注意以下几点问题：

- VC6 是运行于 Apache 服务器
- 'Thread safe'（线程安全）是运行在Apache上以模块的PHP上，如果你以CGI的模式运行PHP，请选择非线程安全模式（' non-thread safe'）。
- VC9是运行于 IIS 服务器上。
- 下载完你需要的二进制包后，解压压缩包，将'php_mongo.dll'文件添加到你的PHP扩展目录中（ext）。ext目录通常在PHP安装目录下的ext目录。

打开php配置文件 php.ini 添加以下配置：

```
extension=php_mongo.dll
```

重启服务器。

通过浏览器访问phpinfo，如果安装成功，就会看到类型以下的信息：

**mongo**

| MongoDB Support | enabled |
|---|---|
| Version | 1.2.0- |

| Directive | Local Value | Master Value |
|---|---|---|
| mongo.allow_empty_keys | 0 | 0 |
| mongo.allow_persistent | 1 | 1 |
| mongo.auto_reconnect | 1 | 1 |
| mongo.chunk_size | 262144 | 262144 |
| mongo.cmd | $ | $ |
| mongo.default_host | localhost | localhost |
| mongo.default_port | 27017 | 27017 |
| mongo.long_as_object | 0 | 0 |
| mongo.native_long | 0 | 0 |
| mongo.no_id | 0 | 0 |
| mongo.utf8 | 1 | 1 |

## MAC中安装MongoDB PHP扩展驱动

你可以使用'autoconf'安装MongoDB PHP扩展驱动。

你可以使用'Xcode'安装MongoDB PHP扩展驱动。

如果你使用 XAMPP，你可以使用以下命令安装MongoDB PHP扩展驱动：

```
sudo /Applications/XAMPP/xamppfiles/bin/pecl install mongo
```

如果以上命令在XMPP或者MAMP中不起作用，你需要在Github上下载兼容的预编译包。

然后添加 'extension=mongo.so'配置到你的php.ini文件中。

# MongoDB 数据插入

## 描述

本章节中我们将向大家介绍如何将数据插入到MongoDB的集合中。

文档的数据结构和JSON基本一样。

所有存储在集合中的数据都是BSON格式。

BSON是一种类json的一种二进制形式的存储格式,简称Binary JSON

。

## MongoDB数据库切换

以下命令可以使用"myinfo"数据库：

```
use myinfo switch to db myinfo
```

```
> use myinfo
switched to db myinfo
>
```

## 为MongoDB数据库定义一个文档

以下文档可以存储在MongoDB中：

```
document=({"user_id" : "ABCDBWN","password" :"ABCDBWN" ,"date_of_join" :
"15/10/2010" ,"education" :"B.C.A." , "profession" : "DEVELOPER","interest" :
"MUSIC","community_name" :["MODERN MUSIC", "CLASSICAL
MUSIC","WESTERN MUSIC"],"community_moder_id" : ["MR. BBB","MR. JJJ","MR
MMM"],"community_members" : [500,200,1500],"friends_id" :
["MMM123","NNN123","OOO123"],"ban_friends_id" :
["BAN123","BAN456","BAN789"]});
```

命令执行如下图所示：

```
> document=({"user_id" : "ABCDBWN","password" :"ABCDBWN" ,"date_of_join" : "15/1
0/2010" ,"education" :"B.C.A." , "profession" : "DEVELOPER","interest" : "MUSIC"
,"community_name" :["MODERN MUSIC", "CLASSICAL MUSIC","WESTERN MUSIC"],"communit
y_moder_id" : ["MR. BBB","MR. JJJ","MR MMM"],"community_members" : [500,200,1500
],"friends_id" : ["MMM123","NNN123","OOO123"],"ban_friends_id" :["BAN123","BAN45
6","BAN789"]});
```

## 显示已定义的文档

已定义的文档显示格式如下所示：

```
{
        "user_id" : "ABCDBWN",
        "password" : "ABCDBWN",
        "date_of_join" : "15/10/2010",
        "education" : "B.C.A.",
        "profession" : "DEVELOPER",
        "interest" : "MUSIC",
        "community_name" : [
                "MODERN MUSIC",
                "CLASSICAL MUSIC",
                "WESTERN MUSIC"
        ],
        "community_moder_id" : [
                "MR. BBB",
                "MR. JJJ",
                "MR MMM"
        ],
        "community_members" : [
                500,
                200,
                1500
        ],
        "friends_id" : [
                "MMM123",
                "NNN123",
                "OOO123"
        ],
        "ban_friends_id" : [
                "BAN123",
                "BAN456",
                "BAN789"
        ]
}
```

# 在集合中插入文档

将以上的文档数据存储到"myinfo" 数据库中的 "userdetails" 集合，执行如下命令：

```
db.userdetails.insert(document)
```

```
>
> db.userdetails.insert(document);
> _
```

# 使用换行符插入数据

当文档的数据较多的时候，我们可以使用换行符来分割文档数据，如下所示：

```
document=({"user_id" : "ABCDBWN","password" :"ABCDBWN" ,"date_of_join" : "15/10/2010" ,
"education" :"B.C.A." , "profession" : "DEVELOPER","interest" : "MUSIC",
"community_name" :["MODERN MUSIC", "CLASSICAL MUSIC","WESTERN MUSIC"],
"community_moder_id" : ["MR. BBB","MR. JJJ","MR MMM"],
"community_members" : [500,200,1500],"friends_id" : ["MMM123","NNN123","OOO123"],
"ban_friends_id" :["BAN123","BAN456","BAN789"]});
```

命令执行如下图所示：

```
> document=({"user_id" : "ABCDBWN","password" :"ABCDBWN" ,"date_of_join" : "15/1
0/2010" ,
... "education" :"B.C.A." , "profession" : "DEVELOPER","interest" : "MUSIC",
... "community_name" :["MODERN MUSIC", "CLASSICAL MUSIC","WESTERN MUSIC"],
... "community_moder_id" : ["MR. BBB","MR. JJJ","MR MMM"],
... "community_members" : [500,200,1500],"friends_id" : ["MMM123","NNN123","OOO1
23"],
... "ban_friends_id" :["BAN123","BAN456","BAN789"]});
```

# 集合中直接插入数据（无定义文档）

数据可以不用定义文档通过shell直接插入：

```
db.userdetails.insert({"user_id" : "xyz123","password" :"xyz123" ,"date_of_join" : "15/08
"education" :"M.C.A." , "profession" : "Software consultant","interest" : "Film",
"community" : [
{
"name" : "DDD FILM CLUB",
"moder_id" : "MR. DBNA",
"members" : "25000",
},
{
"name" : "AXN MOVIES",
"moder_id" : "DOGLUS HUNT",
"members" : "15000",
},
{
"name" : "UROPEAN FILM LOVERS",
"moder_id" : "AMANT LUIS",
"members" : "20000",
}
],
"friends" :[
{
"user_id" : "KKK258",
},
{
"user_id" : "LLL147",
},
{
"user_id" : "MMM369",
}
],
"ban_friends" :[
{
"user_id" : "BAN147"
},
{
"user_id" : "BAN258"
},
{
"user_id" : "BAN369"
}
]
});
```

命令执行如下图所示：

```
>
> db.userdetails.insert({"user_id" : "xyz123","password" :"xyz123" ,"date_of_joi
n" : "15/08/2010" ,
... "education" :"M.C.A." , "profession" : "Software consultant","interest" : "F
ilm",
... "community" : [
... {
... "name" : "DDD FILM CLUB",
... "moder_id" : "MR. DBNA",
... "members" : "25000",
... },
... {
... "name" : "AXN MOVIES",
... "moder_id" : "DOGLUS HUNT",
... "members" : "15000",
... },
... {
... "name" : "UROPEAN FILM LOVERS",
... "moder_id" : "AMANT LUIS",
... "members" : "20000",
... }
... ],
... "friends" :[
... {
... "user_id" : "KKK258",
... },
... {
... "user_id" : "LLL147",
... },
... {
... "user_id" : "MMM369",
... }
... ],
... "ban_friends" :[
... {
... "user_id" : "BAN147"
... },
... {
... "user_id" : "BAN258"
... },
... {
... "user_id" : "BAN369"
... }
... ]
... });
>
>
```

# 查看集合中的数据

使用以下命令查看集合中的数据：

```
db.userdetails.find();
```

```
{
        "user_id" : "xyz123",
        "password" : "xyz123",
        "date_of_join" : "15/08/2010",
        "education" : "M.C.A.",
        "profession" : "Software consultant",
        "interest" : "Film",
        "community" : [
                {
                        "name" : "DDD FILM CLUB",
                        "moder_id" : "MR. DBNA",
                        "members" : "25000"
                },
                {
                        "name" : "AXN MOVIES",
                        "moder_id" : "DOGLUS HUNT",
                        "members" : "15000"
                },
                {
                        "name" : "UROPEAN FILM LOVERS",
                        "moder_id" : "AMANT LUIS",
                        "members" : "20000"
                }
        ],
        "friends" : [
                {
                        "user_id" : "KKK258"
                },
                {
                        "user_id" : "LLL147"
                },
                {
                        "user_id" : "MMM369"
                }
        ],
        "ban_friends" : [
                {
                        "user_id" : "BAN147"
                },
                {
                        "user_id" : "BAN258"
                },
                {
                        "user_id" : "BAN369"
                }
        ]
}
```

# MongoDB使用update()函数更新数据

## 描述

本章节我们将开始学习如何更新MongoDB中的集合数据。

MongoDB数据更新可以使用update()函数。

```
db.collection.update( criteria, objNew, upsert, multi )
```

update()函数接受以下四个参数：

- **criteria** : update的查询条件，类似sql update查询内where后面的。
- **objNew** : update的对象和一些更新的操作符（如$,$inc...）等，也可以理解为sql update查询内set后面的
- **upsert** : 这个参数的意思是，如果不存在update的记录，是否插入objNew,true为插入，默认是false，不插入。
- **multi** : mongodb默认是false,只更新找到的第一条记录，如果这个参数为true,就把按条件查出来多条记录全部更新。

在本教程中我们使用的数据库名称为"myinfo"，集合名称为"userdetails"，以下为插入的数据：

```
> document=({"user_id" : "MNOPBWN","password" :"MNOPBWN" ,"date_of_join" : "16/10/2010"
,"education" :"M.C.A." , "profession" : "CONSULTANT","interest" : "MUSIC","community_name
"CLASSICAL MUSIC","WESTERN MUSIC"],"community_moder_id" : ["MR. BBB","MR. JJJ","MR MMM"],
[500,200,1500],"friends_id" : ["MMM123","NNN123","OOO123"],"ban_friends_id" :["BAN123","B
```

```
> db.userdetails.insert(document)
```

```
> document=({"user_id" : "QRSTBWN","password" :"QRSTBWN" ,"date_of_join" : "17/10/2010" ,
, "profession" : "MARKETING","interest" : "MUSIC","community_name" :["MODERN MUSIC", "CLA
MUSIC"],"community_moder_id" : ["MR. BBB","MR. JJJ","MR MMM"],"community_members" : [500,
["MMM123","NNN123","OOO123"],"ban_friends_id" :["BAN123","BAN456","BAN789"]});
```

```
> db.userdetails.insert(document)
```

## update() 命令

如果我们想将"userdetails"集合中"user_id"为"QRSTBWN"的"password"字段修改为"NEWPASSWORD"，那么我们可以使用update()命令来实现（如下实例所示）。

如果criteria参数匹配集合中的任何一条数据，它将会执行替换命令，否则会插入一条新的数据。

以下实例将更新第一条匹配条件的数据：

```
> db.userdetails.update({"user_id" : "QRSTBWN"},{"user_id" : "QRSTBWN","password" :"NEWPA
,"date_of_join" : "17/10/2010" ,"education" :"M.B.A." , "profession" : "MARKETING","inter
"MUSIC","community_name" :["MODERN MUSIC", "CLASSICAL MUSIC","WESTERN MUSIC"],"community_
BBB","MR. JJJ","MR MMM"],"community_members" : [500,200,1500],"friends_id" : ["MMM123","N
```

```
>
> db.userdetails.update({"user_id" : "QRSTBWN"},{"user_id" : "QRSTBWN","password
" :"NEWPASSWORD" ,"date_of_join" : "17/10/2010" ,"education" :"M.B.A." , "profes
sion" : "MARKETING","interest" : "MUSIC","community_name" :["MODERN MUSIC", "CLA
SSICAL MUSIC","WESTERN MUSIC"],"community_moder_id" : ["MR. BBB","MR. JJJ","MR M
MM"],"community_members" : [500,200,1500],"friends_id" : ["MMM123","NNN123","OOO
123"],"ban_friends_id" :["BAN123","BAN456","BAN789"]});
>
>
```

# 查看集合中更新后的数据

我们可以使用以下命令查看数据是否更新：

```
> db.userdetails.find();
```

```
] }
{ "_id" : ObjectId("4e26ca5f54e3b76638e4dbf6"), "user_id" : "MNOPBWN", "password
" : "MNOPBWN", "date_of_join" : "16/10/2010", "education" : "M.C.A.", "professio
n" : "CONSULTANT", "interest" : "MUSIC", "community_name" : [ "MODERN MUSIC", "C
LASSICAL MUSIC", "WESTERN MUSIC" ], "community_moder_id" : [ "MR. BBB", "MR. JJJ
", "MR MMM" ], "community_members" : [ 500, 200, 1500 ], "friends_id" : [ "MMM12
3", "NNN123", "OOO123" ], "ban_friends_id" : [ "BAN123", "BAN456", "BAN789" ] }
{ "_id" : ObjectId("4e26ca8754e3b76638e4dbf7"), "user_id" : "QRSTBWN", "password
" : "NEWPASSWORD", "date_of_join" : "17/10/2010", "education" : "M.B.A.", "profe
ssion" : "MARKETING", "interest" : "MUSIC", "community_name" : [ "MODERN MUSIC",
 "CLASSICAL MUSIC", "WESTERN MUSIC" ], "community_moder_id" : [ "MR. BBB", "MR.
JJJ", "MR MMM" ], "community_members" : [ 500, 200, 1500 ], "friends_id" : [ "MM
M123", "NNN123", "OOO123" ], "ban_friends_id" : [ "BAN123", "BAN456", "BAN789" ]
 }
>
>
```

# 更多实例

只更新第一条记录：

```
db.test0.update( { "count" : { $gt : 1 } } , { $set : { "test2" : "OK"} } );
```

全部更新：

```
db.test0.update( { "count" : { $gt : 3 } } , { $set : { "test2" : "OK"} },false,true );
```

只添加第一条：

```
db.test0.update( { "count" : { $gt : 4 } } , { $set : { "test5" : "OK"} },true,false );
```

全部添加加进去:

```
db.test0.update( { "count" : { $gt : 5 } } , { $set : { "test5" : "OK"} },true,true );
```

全部更新：

```
db.test0.update( { "count" : { $gt : 15 } } , { $inc : { "count" : 1} },false,true );
```

只更新第一条记录：

```
db.test0.update( { "count" : { $gt : 10 } } , { $inc : { "count" : 1} },false,false );
```

# MongoDB使用- remove()函数删除数据

## 描述

在前面的几个章节中我们已经学习了MongoDB中如何为集合添加数据和更新数据。在本章节中我们将继续学习MongoDB集合的删除。

MongoDB remove()函数是用来移除集合中的数据。

MongoDB数据更新可以使用update()函数。在执行remove()函数前先执行find()命令来判断执行的条件是否正确，这是一个比较好的习惯。

我们使用的数据库名称为**"myinfo"** 我们的集合名称为**"userdetails"**，以下为我们插入的数据：

```
> document=({"user_id" : "testuser","password" :"testpassword" ,"date_of_join" : "16/10/2
:"M.C.A." , "profession" : "CONSULTANT","interest" : "MUSIC","community_name" :["MODERN M
MUSIC","WESTERN MUSIC"],"community_moder_id" : ["MR. BBB","MR. JJJ","MR MMM"],"community_
[500,200,1500],"friends_id" : ["MMM123","NNN123","OOO123"],"ban_friends_id" :["BAN123","B
```

```
> db.userdetails.insert(document)
```

## 查看集合中已经插入的数据

```
> db.userdetails.find();
```

```
        }
] }
{ "_id" : ObjectId("4e26ca5f54e3b76638e4dbf6"), "user_id" : "MNOPBWN", "password
" : "MNOPBWN", "date_of_join" : "16/10/2010", "education" : "M.C.A.", "professio
n" : "CONSULTANT", "interest" : "MUSIC", "community_name" : [ "MODERN MUSIC", "C
LASSICAL MUSIC", "WESTERN MUSIC" ], "community_moder_id" : [ "MR. BBB", "MR. JJJ
", "MR MMM" ], "community_members" : [ 500, 200, 1500 ], "friends_id" : [ "MMM12
3", "NNN123", "OOO123" ], "ban_friends_id" : [ "BAN123", "BAN456", "BAN789" ] }
{ "_id" : ObjectId("4e26ca8754e3b76638e4dbf7"), "user_id" : "QRSTBWN", "password
" : "NEWPASSWORD", "date_of_join" : "17/10/2010", "education" : "M.B.A.", "profe
ssion" : "MARKETING", "interest" : "MUSIC", "community_name" : [ "MODERN MUSIC",
 "CLASSICAL MUSIC", "WESTERN MUSIC" ], "community_moder_id" : [ "MR. BBB", "MR.
JJJ", "MR MMM" ], "community_members" : [ 500, 200, 1500 ], "friends_id" : [ "MM
M123", "NNN123", "OOO123" ], "ban_friends_id" : [ "BAN123", "BAN456", "BAN789" ]
 }
{ "_id" : ObjectId("4e327b4bb94947d1cab92065"), "user_id" : "testuser", "passwor
d" : "testpassword", "date_of_join" : "16/10/2010", "education" : "M.C.A.", "pro
fession" : "CONSULTANT", "interest" : "MUSIC", "community_name" : [ "MODERN MUSI
C", "CLASSICAL MUSIC", "WESTERN MUSIC" ], "community_moder_id" : [ "MR. BBB", "M
R. JJJ", "MR MMM" ], "community_members" : [ 500, 200, 1500 ], "friends_id" : [
"MMM123", "NNN123", "OOO123" ], "ban_friends_id" : [ "BAN123", "BAN456", "BAN789
" ] }
>
```

# 使用 remove() 函数移除数据

如果你想移除"userdetails"集合中"user_id" 为 "testuser"的数据你可以执行以下命令：

```
> db.userdetails.remove( { "user_id" : "testuser" } )
```

# 删除所有数据

如果你想删除"userdetails"集合中的所有数据，可以执行以下命令：

```
> db.userdetails.remove({})
```

# 使用drop()删除集合

如果你想删除整个"userdetails"集合，包含所有文档数据，可以执行以下数据：

```
> db.userdetails.drop()
```

```
>
> db.userdetails.drop()
true
>
>
> _
```

drop()函数返回 true或者false。以上执行结果返回了true，说明操作成功。

# 使用dropDatabase()函数删除数据库

如果你想删除整个数据库的数据，你可以执行以下命令：

```
> db.dropDatabase()
```

执行命令前查看当前使用的数据库是一个良好的习惯，这样可以确保你要删除数据库是正确的，以免造成误操作而产生数据丢失的后果：

```
>
> db
myinfo
>
>
```

```
>
> db.dropDatabase()
{ "dropped" : "myinfo", "ok" : 1 }
> _
```

# MongoDB 查询

## 描述

本教程我们将向大家介绍如何在MongoDB集合中获取数据。

我们使用的数据库名称为**"myinfo"** 我们的集合名称为**"userdetails"**，以下为我们插入的数据：

```
> db.userdetails.insert({"user_id" : "user1","password" :"1a2b3c" ,"date_of_join" : "16/1
, "profession" : "CONSULTANT","interest" : "MUSIC","community_name" :["MODERN MUSIC", "CL
MUSIC"],"community_moder_id" : ["MR. Alex","MR. Dang","MR Haris"],"community_members" :
[700,200,1500],"friends_id" : ["kumar","harry","anand"],"ban_friends_id" :["Amir","Raja",
```

```
> db.userdetails.insert({"user_id" : "user2","password" :"11aa1a" ,"date_of_join" : "17/1
:"M.B.A." , "profession" : "MARKETING","interest" : "MUSIC","community_name" :["MODERN MU
MUSIC","WESTERN MUSIC"],"community_moder_id" : ["MR. Roy","MR. Das","MR Doglus"],"communi
[500,300,1400],"friends_id" : ["pal","viki","john"],"ban_friends_id" :["jalan","monoj","e
```

```
> db.userdetails.insert({"user_id" : "user3","password" :"b1c1d1" ,"date_of_join" : "16/1
:"M.C.A." , "profession" : "IT COR.","interest" : "ART","community_name" :["MODERN ART",
ART"],"community_moder_id" : ["MR. Rifel","MR. Sarma","MR Bhatia"],"community_members" :
[5000,2000,1500],"friends_id" : ["philip","anant","alan"],"ban_friends_id" :["Amir","Raja
```

```
> db.userdetails.insert({"user_id" : "user4","password" :"abczyx" ,"date_of_join" : "17/8
:"M.B.B.S." , "profession" : "DOCTOR","interest" : "SPORTS","community_name" :["ATHELATIC
GYES","FAVOURIT GAMES"],"community_moder_id" : ["MR. Paul","MR. Das","MR Doglus"],"commun
[2500,2200,3500],"friends_id" : ["vinod","viki","john"],"ban_friends_id" :["jalan","monoj
```

## 从集合中获取数据

如果你想在集合中读取所有的的数据，可以执行以下命令

```
> db.userdetails.find();
```

类似于如下SQL查询语句：

```
Select * from userdetails;
```

输出数据如下所示：

```
>
> db.userdetails.find()
{ "_id" : ObjectId("4e367fd6449f200f44a29f00"), "user_id" : "user1", "password"
: "1a2b3c", "date_of_join" : "16/10/2010", "education" : "M.C.A.", "profession"
: "CONSULTANT", "interest" : "MUSIC", "community_name" : [ "MODERN MUSIC", "CLAS
SICAL MUSIC", "WESTERN MUSIC" ], "community_moder_id" : [ "MR. Alex", "MR. Dang"
, "MR Haris" ], "community_members" : [ 700, 200, 1500 ], "friends_id" : [ "kuma
r", "harry", "anand" ], "ban_friends_id" : [ "Amir", "Raja", "mont" ] }
{ "_id" : ObjectId("4e367feb449f200f44a29f01"), "user_id" : "user2", "password"
: "11aa1a", "date_of_join" : "17/10/2009", "education" : "M.B.A.", "profession"
: "MARKETING", "interest" : "MUSIC", "community_name" : [ "MODERN MUSIC", "CLASS
ICAL MUSIC", "WESTERN MUSIC" ], "community_moder_id" : [ "MR. Roy", "MR. Das", "
MR Doglus" ], "community_members" : [ 500, 300, 1400 ], "friends_id" : [ "pal",
"viki", "john" ], "ban_friends_id" : [ "jalan", "monoj", "evan" ] }
{ "_id" : ObjectId("4e367ff7449f200f44a29f02"), "user_id" : "user3", "password"
: "b1c1d1", "date_of_join" : "16/10/2010", "education" : "M.C.A.", "profession"
: "IT COR.", "interest" : "ART", "community_name" : [ "MODERN ART", "CLASSICAL A
RT", "WESTERN ART" ], "community_moder_id" : [ "MR. Rifel", "MR. Sarma", "MR Bha
tia" ], "community_members" : [ 5000, 2000, 1500 ], "friends_id" : [ "philip", "
anant", "alan" ], "ban_friends_id" : [ "Amir", "Raja", "mont" ] }
{ "_id" : ObjectId("4e368006449f200f44a29f03"), "user_id" : "user4", "password"
: "abczyx", "date_of_join" : "17/8/2009", "education" : "M.B.B.S.", "profession"
 : "DOCTOR", "interest" : "SPORTS", "community_name" : [ "ATHELATIC", "GAMES FAN
 GYES", "FAVOURIT GAMES" ], "community_moder_id" : [ "MR. Paul", "MR. Das", "MR
Doglus" ], "community_members" : [ 2500, 2200, 3500 ], "friends_id" : [ "vinod",
 "viki", "john" ], "ban_friends_id" : [ "jalan", "monoj", "evan" ] }
>
```

# 通过指定条件读取数据

如果我们想在集合"userdetails"中读取"education"为"M.C.A." 的数据，我们可以执行以下命令：

```
> db.userdetails.find({"education":"M.C.A."})
```

类似如下SQL查询语句:

```
Select * from userdetails where education="M.C.A.";
```

输出结果如下所示：

```
>
> db.userdetails.find({"education":"M.C.A."})
{ "_id" : ObjectId("4e367fd6449f200f44a29f00"), "user_id" : "user1", "password"
: "1a2b3c", "date_of_join" : "16/10/2010", "education" : "M.C.A.", "profession"
: "CONSULTANT", "interest" : "MUSIC", "community_name" : [ "MODERN MUSIC", "CLAS
SICAL MUSIC", "WESTERN MUSIC" ], "community_moder_id" : [ "MR. Alex", "MR. Dang"
, "MR Haris" ], "community_members" : [ 700, 200, 1500 ], "friends_id" : [ "kuma
r", "harry", "anand" ], "ban_friends_id" : [ "Amir", "Raja", "mont" ] }
{ "_id" : ObjectId("4e367ff7449f200f44a29f02"), "user_id" : "user3", "password"
: "b1c1d1", "date_of_join" : "16/10/2010", "education" : "M.C.A.", "profession"
: "IT COR.", "interest" : "ART", "community_name" : [ "MODERN ART", "CLASSICAL A
RT", "WESTERN ART" ], "community_moder_id" : [ "MR. Rifel", "MR. Sarma", "MR Bha
tia" ], "community_members" : [ 5000, 2000, 1500 ], "friends_id" : [ "philip", "
anant", "alan" ], "ban_friends_id" : [ "Amir", "Raja", "mont" ] }
> _
```

# MongoDB条件操作符

## 描述

条件操作符用于比较两个表达式并从mongoDB集合中获取数据。

在本章节中，我们将讨论如何在MongoDB中使用条件操作符。

MongoDB中条件操作符有：

- (>) 大于 - $gt
- (<) 小于 - $lt
- (>=) 大于等于 - $gte
- (<= ) 小于等于 - $lte

我们使用的数据库名称为**"myinfo"** 我们的集合名称为**"testtable"**，以下为我们插入的数据。

简单的集合"testtable"：

```
>
> db.testtable.find()
{ "_id" : ObjectId("4e3d1ee9f52b9bd78374f952"), "user_id" : "user1", "password"
: "1a2b3c", "sex" : "Male", "age" : 17, "date_of_join" : "16/10/2010", "educatio
n" : "M.C.A.", "profession" : "CONSULTANT", "interest" : "MUSIC", "extra" : { "c
ommunity_name" : [ "MODERN MUSIC", "CLASSICAL MUSIC", "WESTERN MUSIC" ], "commun
ity_moder_id" : [ "MR. Alex", "MR. Dang", "MR Haris" ], "community_members" : [
700, 200, 1500 ], "friends" : { "valued_friends_id" : [ "kumar", "harry", "anand
" ], "ban_friends_id" : [ "Amir", "Raja", "mont" ] } } }
{ "_id" : ObjectId("4e3d1f1cf52b9bd78374f953"), "user_id" : "user2", "password"
: "11aa1a", "sex" : "Male", "age" : 24, "date_of_join" : "17/10/2009", "educatio
n" : "M.B.A.", "profession" : "MARKETING", "interest" : "MUSIC", "extra" : { "co
mmunity_name" : [ "MODERN MUSIC", "CLASSICAL MUSIC", "WESTERN MUSIC" ], "communi
ty_moder_id" : [ "MR. Roy", "MR. Das", "MR Doglus" ], "community_members" : [ 50
0, 300, 1400 ], "friends" : { "valuled_friends_id" : [ "pal", "viki", "john" ],
"ban_friends_id" : [ "jalan", "monoj", "evan" ] } } }
{ "_id" : ObjectId("4e3d1f2af52b9bd78374f954"), "user_id" : "user3", "password"
: "b1c1d1", "sex" : "Female", "age" : 19, "date_of_join" : "16/10/2010", "educat
ion" : "M.C.A.", "profession" : "IT COR.", "interest" : "ART", "extra" : { "comm
unity_name" : [ "MODERN ART", "CLASSICAL ART", "WESTERN ART" ], "community_moder
_id" : [ "MR. Rifel", "MR. Sarma", "MR Bhatia" ], "community_members" : [ 5000,
2000, 1500 ], "friends" : { "valued_friends_id" : [ "philip", "anant", "alan" ],
 "ban_friends_id" : [ "Amir", "Raja", "mont" ] } } }
{ "_id" : ObjectId("4e3d1f35f52b9bd78374f955"), "user_id" : "user4", "password"
: "abczyx", "sex" : "Female", "age" : 22, "date_of_join" : "17/8/2009", "educati
on" : "M.B.B.S.", "profession" : "DOCTOR", "interest" : "SPORTS", "extra" : { "c
ommunity_name" : [ "ATHELATIC", "GAMES FAN GYES", "FAVOURIT GAMES" ], "community
_moder_id" : [ "MR. Paul", "MR. Das", "MR Doglus" ], "community_members" : [ 250
0, 2200, 3500 ], "friends" : { "valued_friends_id" : [ "vinod", "viki", "john" ]
, "ban_friends_id" : [ "jalan", "monoj", "evan" ] } } }
>
>
```

## MongoDB (>) 大于操作符 - $gt

如果你想获取"testtable"集合中"age" 大于22的数据，你可以使用以下命令：

```
> db.testtable.find({age : {$gt : 22}})
```

类似于SQL语句：

```
Select * from testtable where age >22;
```

输出结果：

```
>
> db.testtable.find({age : {$gt : 22}})
{ "_id" : ObjectId("4e3d1f1cf52b9bd78374f953"), "user_id" : "user2", "password"
: "11aa1a", "sex" : "Male", "age" : 24, "date_of_join" : "17/10/2009", "educatio
n" : "M.B.A.", "profession" : "MARKETING", "interest" : "MUSIC", "extra" : { "co
mmunity_name" : [ "MODERN MUSIC", "CLASSICAL MUSIC", "WESTERN MUSIC" ], "communi
ty_moder_id" : [ "MR. Roy", "MR. Das", "MR Doglus" ], "community_members" : [ 50
0, 300, 1400 ], "friends" : { "valuled_friends_id" : [ "pal", "viki", "john" ],
"ban_friends_id" : [ "jalan", "monoj", "evan" ] } } }
>
> _
```

# MongoDB（>=）大于等于操作符 - $gte

如果你想获取"testtable"集合中"age" 大于等于22的数据，你可以执行以下命令:

```
> db.testtable.find({age : {$gte : 22}})
```

类似于SQL语句：

```
Select * from testtable where age >=22;
```

输出结果：

```
>
> db.testtable.find({age : {$gte : 22}})
{ "_id" : ObjectId("4e3d1f1cf52b9bd78374f953"), "user_id" : "user2", "password"
: "11aa1a", "sex" : "Male", "age" : 24, "date_of_join" : "17/10/2009", "educatio
n" : "M.B.A.", "profession" : "MARKETING", "interest" : "MUSIC", "extra" : { "co
mmunity_name" : [ "MODERN MUSIC", "CLASSICAL MUSIC", "WESTERN MUSIC" ], "communi
ty_moder_id" : [ "MR. Roy", "MR. Das", "MR Doglus" ], "community_members" : [ 50
0, 300, 1400 ], "friends" : { "valuled_friends_id" : [ "pal", "viki", "john" ],
"ban_friends_id" : [ "jalan", "monoj", "evan" ] } } }
{ "_id" : ObjectId("4e3d1f35f52b9bd78374f955"), "user_id" : "user4", "password"
: "abczyx", "sex" : "Female", "age" : 22, "date_of_join" : "17/8/2009", "educati
on" : "M.B.B.S.", "profession" : "DOCTOR", "interest" : "SPORTS", "extra" : { "c
ommunity_name" : [ "ATHELATIC", "GAMES FAN GYES", "FAVOURIT GAMES" ], "community
_moder_id" : [ "MR. Paul", "MR. Das", "MR Doglus" ], "community_members" : [ 250
0, 2200, 3500 ], "friends" : { "valued_friends_id" : [ "vinod", "viki", "john" ]
, "ban_friends_id" : [ "jalan", "monoj", "evan" ] } } }
>
>
```

# MongoDB (<) 小于操作符 - $lt

如果你想获取"testtable"集合中"age" 小于19的数据，你可以执行以下命令：

类似于SQL语句：

```
Select * from testtable where age <19;
```

输出结果：

```
>
> db.testtable.find({age : {$lt :19}})
{ "_id" : ObjectId("4e3d1ee9f52b9bd78374f952"), "user_id" : "user1", "password"
: "1a2b3c", "sex" : "Male", "age" : 17, "date_of_join" : "16/10/2010", "educatio
n" : "M.C.A.", "profession" : "CONSULTANT", "interest" : "MUSIC", "extra" : { "c
ommunity_name" : [ "MODERN MUSIC", "CLASSICAL MUSIC", "WESTERN MUSIC" ], "commun
ity_moder_id" : [ "MR. Alex", "MR. Dang", "MR Haris" ], "community_members" : [
700, 200, 1500 ], "friends" : { "valued_friends_id" : [ "kumar", "harry", "anand
" ], "ban_friends_id" : [ "Amir", "Raja", "mont" ] } } }
>
>
```

# MongoDB (<=) 小于操作符 - $lte

如果你想获取"testtable"集合中"age" 小于等于19的数据，你可以执行以下命令：

```
> db.testtable.find({age : {$lte : 19}})
```

类似于SQL语句：

```
Select * from testtable where age <=19;
```

输出结果：

```
>
> db.testtable.find({age : {$lte : 19}})
{ "_id" : ObjectId("4e3d1ee9f52b9bd78374f952"), "user_id" : "user1", "password"
: "1a2b3c", "sex" : "Male", "age" : 17, "date_of_join" : "16/10/2010", "educatio
n" : "M.C.A.", "profession" : "CONSULTANT", "interest" : "MUSIC", "extra" : { "c
ommunity_name" : [ "MODERN MUSIC", "CLASSICAL MUSIC", "WESTERN MUSIC" ], "commun
ity_moder_id" : [ "MR. Alex", "MR. Dang", "MR Haris" ], "community_members" : [
700, 200, 1500 ], "friends" : { "valued_friends_id" : [ "kumar", "harry", "anand
" ], "ban_friends_id" : [ "Amir", "Raja", "mont" ] } } }
{ "_id" : ObjectId("4e3d1f2af52b9bd78374f954"), "user_id" : "user3", "password"
: "b1c1d1", "sex" : "Female", "age" : 19, "date_of_join" : "16/10/2010", "educat
ion" : "M.C.A.", "profession" : "IT COR.", "interest" : "ART", "extra" : { "comm
unity_name" : [ "MODERN ART", "CLASSICAL ART", "WESTERN ART" ], "community_moder
_id" : [ "MR. Rifel", "MR. Sarma", "MR Bhatia" ], "community_members" : [ 5000,
2000, 1500 ], "friends" : { "valued_friends_id" : [ "philip", "anant", "alan" ],
 "ban_friends_id" : [ "Amir", "Raja", "mont" ] } } }
>
>
```

# MongoDB 使用 (<) 和 (>) 查询operator - $lt 和 $gt

如果你想获取"testtable"集合中"age" 大于17以及小于24的数据，你可以执行以下命令：

```
> db.testtable.find({age : {$lt :24, $gt : 17}})
```

类似于SQL语句：

```
Select * from testtable where age 17;
```

输出结果：

```
>
> db.testtable.find({age : {$lt :24, $gt : 17}})
{ "_id" : ObjectId("4e3d1f2af52b9bd78374f954"), "user_id" : "user3", "password"
: "b1c1d1", "sex" : "Female", "age" : 19, "date_of_join" : "16/10/2010", "educat
ion" : "M.C.A.", "profession" : "IT COR.", "interest" : "ART", "extra" : { "comm
unity_nane" : [ "MODERN ART", "CLASSICAL ART", "WESTERN ART" ], "community_moder
_id" : [ "MR. Rifel", "MR. Sarma", "MR Bhatia" ], "community_members" : [ 5000,
2000, 1500 ], "friends" : { "valued_friends_id" : [ "philip", "anant", "alan" ],
 "ban_friends_id" : [ "Amir", "Raja", "mont" ] } } }
{ "_id" : ObjectId("4e3d1f35f52b9bd78374f955"), "user_id" : "user4", "password"
: "abczyx", "sex" : "Female", "age" : 22, "date_of_join" : "17/8/2009", "educati
on" : "M.B.B.S.", "profession" : "DOCTOR", "interest" : "SPORTS", "extra" : { "c
ommunity_name" : [ "ATHELATIC", "GAMES FAN GYES", "FAVOURIT GAMES" ], "community
_moder_id" : [ "MR. Paul", "MR. Das", "MR Doglus" ], "community_members" : [ 250
0, 2200, 3500 ], "friends" : { "valued_friends_id" : [ "vinod", "viki", "john" ]
, "ban_friends_id" : [ "jalan", "monoj", "evan" ] } } }
>
> _
```

# MongoDB条件操作符 - $type

## 描述

在本章节中，我们将继续讨论MongoDB中条件操作符 $type。

$type操作符是基于BSON类型来检索集合中匹配的结果。

MongoDB中可以使用的类型：

| 类型描述 | 类型值 |
|---|---|
| Double | 1 |
| String | 2 |
| Object | 3 |
| Array | 4 |
| Binary data | 5 |
| Object id | 7 |
| Boolean | 8 |
| Date | 9 |
| Null | 10 |
| Regular expression | 11 |
| JavaScript code | 13 |
| Symbol | 14 |
| JavaScript code with scope | 15 |
| 32-bit integer | 16 |
| Timestamp | 17 |
| 64-bit integer | 18 |
| Min key | 255 |
| Max key | 127 |

我们使用的数据库名称为**"myinfo"** 我们的集合名称为**"testtable"**，以下为我们插入的数据。

简单的集合"testtable"：

```
>
> db.testtable.find()
{ "_id" : ObjectId("4e3d1ee9f52b9bd78374f952"), "user_id" : "user1", "password"
: "1a2b3c", "sex" : "Male", "age" : 17, "date_of_join" : "16/10/2010", "educatio
n" : "M.C.A.", "profession" : "CONSULTANT", "interest" : "MUSIC", "extra" : { "c
ommunity_name" : [ "MODERN MUSIC", "CLASSICAL MUSIC", "WESTERN MUSIC" ], "commun
ity_moder_id" : [ "MR. Alex", "MR. Dang", "MR Haris" ], "community_members" : [
700, 200, 1500 ], "friends" : { "valued_friends_id" : [ "kumar", "harry", "anand
" ], "ban_friends_id" : [ "Amir", "Raja", "mont" ] } } }
{ "_id" : ObjectId("4e3d1f1cf52b9bd78374f953"), "user_id" : "user2", "password"
: "11aa1a", "sex" : "Male", "age" : 24, "date_of_join" : "17/10/2009", "educatio
n" : "M.B.A.", "profession" : "MARKETING", "interest" : "MUSIC", "extra" : { "co
mmunity_name" : [ "MODERN MUSIC", "CLASSICAL MUSIC", "WESTERN MUSIC" ], "communi
ty_moder_id" : [ "MR. Roy", "MR. Das", "MR Doglus" ], "community_members" : [ 50
0, 300, 1400 ], "friends" : { "valuled_friends_id" : [ "pal", "viki", "john" ],
"ban_friends_id" : [ "jalan", "monoj", "evan" ] } } }
{ "_id" : ObjectId("4e3d1f2af52b9bd78374f954"), "user_id" : "user3", "password"
: "b1c1d1", "sex" : "Female", "age" : 19, "date_of_join" : "16/10/2010", "educat
ion" : "M.C.A.", "profession" : "IT COR.", "interest" : "ART", "extra" : { "comm
unity_name" : [ "MODERN ART", "CLASSICAL ART", "WESTERN ART" ], "community_moder
_id" : [ "MR. Rifel", "MR. Sarma", "MR Bhatia" ], "community_members" : [ 5000,
2000, 1500 ], "friends" : { "valued_friends_id" : [ "philip", "anant", "alan" ],
 "ban_friends_id" : [ "Amir", "Raja", "mont" ] } } }
{ "_id" : ObjectId("4e3d1f35f52b9bd78374f955"), "user_id" : "user4", "password"
: "abczyx", "sex" : "Female", "age" : 22, "date_of_join" : "17/8/2009", "educati
on" : "M.B.B.S.", "profession" : "DOCTOR", "interest" : "SPORTS", "extra" : { "c
ommunity_name" : [ "ATHELATIC", "GAMES FAN GYES", "FAVOURIT GAMES" ], "community
_moder_id" : [ "MR. Paul", "MR. Das", "MR Doglus" ], "community_members" : [ 250
0, 2200, 3500 ], "friends" : { "valued_friends_id" : [ "vinod", "viki", "john" ]
, "ban_friends_id" : [ "jalan", "monoj", "evan" ] } } }
{ "_id" : ObjectId("4e578544d10a10ad4b593f24"), "user_id" : "user5", "password"
: "user5", "sex" : "Male", "age" : 21, "date_of_join" : "17/08/2011", "education
" : "MCA", "profession" : "S.W. Engineer", "interest" : "SPORTS", "extra" : { "c
ommunity_name" : [ "ATHELATIC", "GAMES FAN GYES", "FAVOURIT GAMES" ] } }
>
>
```

# MongoDB 操作符 - $type 实例

如果想获取 "testtable" 集合包含在 "extra" 中的"friends"为BSON类型的对象，你可以使用以下命令：

```
> db.testtable.find({"extra.friends" : {$type : 3}})
```

```
>
> db.testtable.find({"extra.friends" : {$type : 3}})
{ "_id" : ObjectId("4e3d1ee9f52b9bd78374f952"), "user_id" : "user1", "password"
: "1a2b3c", "sex" : "Male", "age" : 17, "date_of_join" : "16/10/2010", "educatio
n" : "M.C.A.", "profession" : "CONSULTANT", "interest" : "MUSIC", "extra" : { "c
ommunity_name" : [ "MODERN MUSIC", "CLASSICAL MUSIC", "WESTERN MUSIC" ], "commun
ity_moder_id" : [ "MR. Alex", "MR. Dang", "MR Haris" ], "community_members" : [
700, 200, 1500 ], "friends" : { "valued_friends_id" : [ "kumar", "harry", "anand
" ], "ban_friends_id" : [ "Amir", "Raja", "mont" ] } } }
{ "_id" : ObjectId("4e3d1f1cf52b9bd78374f953"), "user_id" : "user2", "password"
: "11aa1a", "sex" : "Male", "age" : 24, "date_of_join" : "17/10/2009", "educatio
n" : "M.B.A.", "profession" : "MARKETING", "interest" : "MUSIC", "extra" : { "co
mmunity_name" : [ "MODERN MUSIC", "CLASSICAL MUSIC", "WESTERN MUSIC" ], "communi
ty_moder_id" : [ "MR. Roy", "MR. Das", "MR Doglus" ], "community_members" : [ 50
0, 300, 1400 ], "friends" : { "valuled_friends_id" : [ "pal", "viki", "john" ],
"ban_friends_id" : [ "jalan", "monoj", "evan" ] } } }
{ "_id" : ObjectId("4e3d1f2af52b9bd78374f954"), "user_id" : "user3", "password"
: "b1c1d1", "sex" : "Female", "age" : 19, "date_of_join" : "16/10/2010", "educat
ion" : "M.C.A.", "profession" : "IT COR.", "interest" : "ART", "extra" : { "comm
unity_name" : [ "MODERN ART", "CLASSICAL ART", "WESTERN ART" ], "community_moder
_id" : [ "MR. Rifel", "MR. Sarma", "MR Bhatia" ], "community_members" : [ 5000,
2000, 1500 ], "friends" : { "valued_friends_id" : [ "philip", "anant", "alan" ],
 "ban_friends_id" : [ "Amir", "Raja", "mont" ] } } }
{ "_id" : ObjectId("4e3d1f35f52b9bd78374f955"), "user_id" : "user4", "password"
: "abczyx", "sex" : "Female", "age" : 22, "date_of_join" : "17/8/2009", "educati
on" : "M.B.B.S.", "profession" : "DOCTOR", "interest" : "SPORTS", "extra" : { "c
ommunity_name" : [ "ATHELATIC", "GAMES FAN GYES", "FAVOURIT GAMES" ], "community
_moder_id" : [ "MR. Paul", "MR. Das", "MR Doglus" ], "community_members" : [ 250
0, 2200, 3500 ], "friends" : { "valued_friends_id" : [ "vinod", "viki", "john" ]
, "ban_friends_id" : [ "jalan", "monoj", "evan" ] } } }
>
>
```

# 更多实例

查询所有name字段是字符类型的数据：

```
db.users.find({name: {$type: 2}});
```

查询所有age字段是整型的数据：

```
db.users.find({age: {$type: 16}});
```

# MongoDB Limit与Skip方法

# MongoDB Limit() 方法

如果你需要在MongoDB中读取指定数量的数据记录，可以使用MongoDB的Limit方法，limit()方法接受一个数字参数，该参数指定从MongoDB中读取的记录条数。

## 语法

limit()方法基本语法如下所示：

```
>db.COLLECTION_NAME.find().limit(NUMBER)
```

## 实例

集合 myycol 中的数据如下：

```
{ "_id" : ObjectId(5983548781331adf45ec5), "title":"MongoDB Overview"}
{ "_id" : ObjectId(5983548781331adf45ec6), "title":"NoSQL Overview"}
{ "_id" : ObjectId(5983548781331adf45ec7), "title":"Tutorials Point Overview"}
```

以上实例为显示查询文档中的两条记录：

```
>db.mycol.find({},{"title":1,_id:0}).limit(2)
{"title":"MongoDB Overview"}
{"title":"NoSQL Overview"}
>
```

注：如果你们有指定limit()方法中的参数则显示集合中的所有数据。

# MongoDB Skip() 方法

我们除了可以使用limit()方法来读取指定数量的数据外，还可以使用skip()方法来跳过指定数量的数据，skip方法同样接受一个数字参数作为跳过的记录条数。

## 语法

skip() 方法脚本语法格式如下：

```
>db.COLLECTION_NAME.find().limit(NUMBER).skip(NUMBER)
```

# 实例

以上实例只会显示第二条文档数据

```
>db.mycol.find({},{"title":1,_id:0}).limit(1).skip(1)
{"title":"NoSQL Overview"}
>
```

注:skip()方法默认参数为 0 。

# MongoDB 排序

## MongoDB sort()方法

在MongoDB中使用使用sort()方法对数据进行排序，sort()方法可以通过参数指定排序的字段，并使用 1 和 -1 来指定排序的方式，其中 1 为升序排序，而-1是用于降序排列。

### 语法

sort()方法基本语法如下所示：

```
>db.COLLECTION_NAME.find().sort({KEY:1})
```

### 实例

myycol 集合中的数据如下：

```
{ "_id" : ObjectId(5983548781331adf45ec5), "title":"MongoDB Overview"}
{ "_id" : ObjectId(5983548781331adf45ec6), "title":"NoSQL Overview"}
{ "_id" : ObjectId(5983548781331adf45ec7), "title":"Tutorials Point Overview"}
```

以下实例演示了 myycol 集合中的数据按字段 title 的降序排序：

```
>db.mycol.find({},{"title":1,_id:0}).sort({"title":-1})
{"title":"Tutorials Point Overview"}
{"title":"NoSQL Overview"}
{"title":"MongoDB Overview"}
>
```

注： 如果没有指定sort()方法的排序方式，默认按照文档的升序排序。

# MongoDB 索引

索引通常能够极大的提高查询的效率，如果没有索引，MongoDB在读取数据时必须扫描集合中的每个文件并选取那些符合查询条件的记录。

这种扫描全集合的查询效率是非常低的，特别在处理大量的数据时，查询可以要花费几十秒甚至几分钟，这对网站的性能是非常致命的。

索引是特殊的数据结构，索引存储在一个易于遍历读取的数据集合中，索引是对数据库表中一列或多列的值进行排序的一种结构

# ensureIndex() 方法

MongoDB使用 ensureIndex() 方法来创建索引。

## 语法

ensureIndex()方法基本语法格式如下所示：

```
>db.COLLECTION_NAME.ensureIndex({KEY:1})
```

语法中 Key 值为你要创建的索引字段，1为指定按升序创建索引，如果你想按降序来创建索引指定为-1即可。

## 实例

```
>db.mycol.ensureIndex({"title":1})
>
```

ensureIndex() 方法中你也可以设置使用多个字段创建索引（关系型数据库中称作复合索引）。

```
>db.mycol.ensureIndex({"title":1,"description":-1})
>
```

ensureIndex() 接收可选参数，可选参数列表如下：

| Parameter | Type | Description |
|---|---|---|
| background | Boolean | 建索引过程会阻塞其它数据库操作，background可指定以后台方式创建索引，即增加 "background" 可选参数。 "background" 默认值为**false**。 |
| unique | Boolean | 建立的索引是否唯一。指定为true创建唯一索引。默认值为**false**. |
| name | string | 索引的名称。如果未指定，MongoDB的通过连接索引的字段名和排序顺序生成一个索引名称。 |
| dropDups | Boolean | 在建立唯一索引时是否删除重复记录,指定 true 创建唯一索引。默认值为 **false**. |
| sparse | Boolean | 对文档中不存在的字段数据不启用索引；这个参数需要特别注意，如果设置为true的话，在索引字段中不会查询出不包含对应字段的文档.。默认值为 **false**. |
| expireAfterSeconds | integer | 指定一个以秒为单位的数值，完成 TTL设定，设定集合的生存时间。 |
| v | index version | 索引的版本号。默认的索引版本取决于mongod创建索引时运行的版本。 |
| weights | document | 索引权重值，数值在 1 到 99,999 之间，表示该索引相对于其他索引字段的得分权重。 |
| default_language | string | 对于文本索引，该参数决定了停用词及词干和词器的规则的列表。 默认为英语 |
| language_override | string | 对于文本索引，该参数指定了包含在文档中的字段名，语言覆盖默认的language，默认值为 language. |

## 实例

在后台创建索引：

```
db.values.ensureIndex({open: 1, close: 1}, {background: true})
```

通过在创建索引时加background:true 的选项，让创建工作在后台执行

# MongoDB 聚合

MongoDB中聚合(aggregate)主要用于处理数据(诸如统计平均值,求和等)，并返回计算后的数据结果。有点类似sql语句中的 count(*)。

# aggregate() 方法

MongoDB中聚合的方法使用aggregate()。

## 语法

aggregate() 方法的基本语法格式如下所示：

```
>db.COLLECTION_NAME.aggregate(AGGREGATE_OPERATION)
```

## 实例

集合中的数据如下：

```
{
   _id: ObjectId(7df78ad8902c)
   title: 'MongoDB Overview',
   description: 'MongoDB is no sql database',
   by_user: 'w3cschool.cc',
   url: 'http://www.w3cschool.cc',
   tags: ['mongodb', 'database', 'NoSQL'],
   likes: 100
},
{
   _id: ObjectId(7df78ad8902d)
   title: 'NoSQL Overview',
   description: 'No sql database is very fast',
   by_user: 'w3cschool.cc',
   url: 'http://www.w3cschool.cc',
   tags: ['mongodb', 'database', 'NoSQL'],
   likes: 10
},
{
   _id: ObjectId(7df78ad8902e)
   title: 'Neo4j Overview',
   description: 'Neo4j is no sql database',
   by_user: 'Neo4j',
   url: 'http://www.neo4j.com',
   tags: ['neo4j', 'database', 'NoSQL'],
   likes: 750
},
```

现在我们通过以上集合计算每个作者所写的文章数，使用aggregate()计算结果如下：

```
> db.mycol.aggregate([{$group : {_id : "$by_user", num_tutorial : {$sum : 1}}}])
{
   "result" : [
      {
         "_id" : "w3cschool.cc",
         "num_tutorial" : 2
      },
      {
         "_id" : "Neo4j",
         "num_tutorial" : 1
      }
   ],
   "ok" : 1
}
>
```

以上实例类似sql语句： *select by_user, count(*) from mycol group by by_user*

在上面的例子中，我们通过字段by_user字段对数据进行分组，并计算by_user字段相同值的总和。

下表展示了一些聚合的表达式:

| 表达式 | 描述 | 实例 |
|---|---|---|
| $sum | 计算总和。 | db.mycol.aggregate([{$group : {_id : "$by_user", num_tutorial : {$sum : "$likes"}}}]) |
| $avg | 计算平均值 | db.mycol.aggregate([{$group : {_id : "$by_user", num_tutorial : {$avg : "$likes"}}}]) |
| $min | 获取集合中所有文档对应值得最小值。 | db.mycol.aggregate([{$group : {_id : "$by_user", num_tutorial : {$min : "$likes"}}}]) |
| $max | 获取集合中所有文档对应值得最大值。 | db.mycol.aggregate([{$group : {_id : "$by_user", num_tutorial : {$max : "$likes"}}}]) |
| $push | 在结果文档中插入值到一个数组中。 | db.mycol.aggregate([{$group : {_id : "$by_user", url : {$push: "$url"}}}]) |
| $addToSet | 在结果文档中插入值到一个数组中，但不创建副本。 | db.mycol.aggregate([{$group : {_id : "$by_user", url : {$addToSet : "$url"}}}]) |
| $first | 根据资源文档的排序获取第一个文档数据。 | db.mycol.aggregate([{$group : {_id : "$by_user", first_url : {$first : "$url"}}}]) |
| $last | 根据资源文档的排序获取最后一个文档数据 | db.mycol.aggregate([{$group : {_id : "$by_user", last_url : {$last : "$url"}}}]) |

# 管道的概念

管道在Unix和Linux中一般用于将当前命令的输出结果作为下一个命令的参数。

MongoDB的聚合管道将MongoDB文档在一个管道处理完毕后将结果传递给下一个管道处理。管道操作是可以重复的。

表达式：处理输入文档并输出。表达式是无状态的，只能用于计算当前聚合管道的文档，不能处理其它的文档。

这里我们介绍一下聚合框架中常用的几个操作：

- $project：修改输入文档的结构。可以用来重命名、增加或删除域，也可以用于创建计算结果以及嵌套文档。
- $match：用于过滤数据，只输出符合条件的文档。$match使用MongoDB的标准查询操作。
- $limit：用来限制MongoDB聚合管道返回的文档数。
- $skip：在聚合管道中跳过指定数量的文档，并返回余下的文档。
- $unwind：将文档中的某一个数组类型字段拆分成多条，每条包含数组中的一个值。
- $group：将集合中的文档分组，可用于统计结果。
- $sort：将输入文档排序后输出。
- $geoNear：输出接近某一地理位置的有序文档。

## 管道操作符实例

1、$project实例

```
db.article.aggregate(
    { $project : {
        title : 1 ,
        author : 1 ,
    }}
 );
```

这样的话结果中就只还有_id,tilte和author三个字段了，默认情况下_id字段是被包含的，如果要想不包含_id话可以这样:

```
db.article.aggregate(
    { $project : {
        _id : 0 ,
        title : 1 ,
        author : 1
    }});
```

2.$match实例

```
db.articles.aggregate( [
                        { $match : { score : { $gt : 70, $lte : 90 } } },
                        { $group: { _id: null, count: { $sum: 1 } } }
                       ] );
```

$match用于获取分数大于70小于或等于90记录，然后将符合条件的记录送到下一阶段$group管道操作符进行处理。

3.$skip实例

```
db.article.aggregate(
    { $skip : 5 });
```

经过$skip管道操作符处理后，前五个文档被"过滤"掉。

# MongoDB 复制（副本集）

MongoDB复制是将数据同步在多个服务器的过程。

复制提供了数据的冗余备份，并在多个服务器上存储数据副本，提高了数据的可用性， 并可以保证数据的安全性。

复制还允许您从硬件故障和服务中断中恢复数据。

## 什么是复制?

- 保障数据的安全性
- 数据高可用性 (24*7)
- 灾难恢复
- 无需停机维护（如备份，重建索引，压缩）
- 分布式读取数据

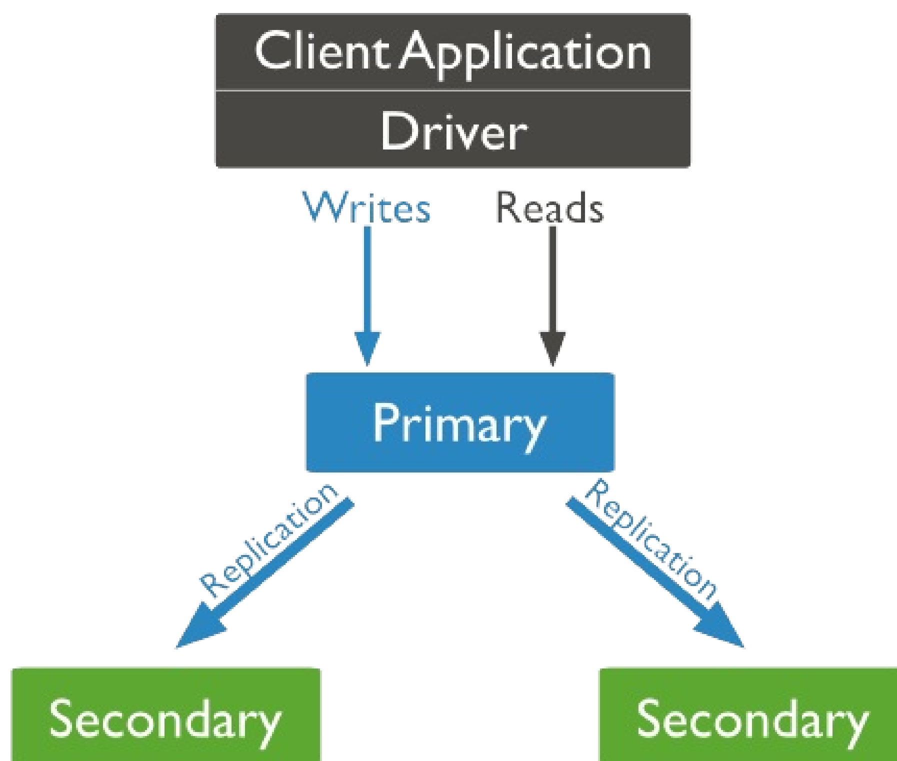
## MongoDB复制原理

mongodb的复制至少需要两个节点。其中一个是主节点，负责处理客户端请求，其余的都是从节点，负责复制主节点上的数据。

mongodb各个节点常见的搭配方式为：一主一从、一主多从。

主节点记录在其上的所有操作oplog，从节点定期轮询主节点获取这些操作，然后对自己的数据副本执行这些操作，从而保证从节点的数据与主节点一致。

MongoDB复制结构图如下所示：




以上结构图总，客户端总主节点读取数据，在客户端写入数据到主节点是， 主节点与从节点进行数据交互保障数据的一致性。

### 副本集特征：

- N 个节点的集群
- 任何节点可作为主节点
- 所有写入操作都在主节点上
- 自动故障转移
- 自动恢复

## MongoDB副本集设置

在本教程中我们使用同一个MongoDB来做MongoDB主从的实验， 操作步骤如下：

1、关闭正在运行的MongoDB服务器。

现在我们通过指定 --replSet 选项来启动mongoDB。--replSet 基本语法格式如下：

```
mongod --port "PORT" --dbpath "YOUR_DB_DATA_PATH" --replSet "REPLICA_SET_INSTANCE_NAME"
```

### 实例

```
mongod --port 27017 --dbpath "D:\set up\mongodb\data" --replSet rs0
```

以上实例会启动一个名为rs0的MongoDB实例，其端口号为27017。

启动后打开命令提示框并连接上mongoDB服务。

在Mongo客户端使用命令rs.initiate()来启动一个新的副本集。

我们可以使用rs.conf()来查看副本集的配置

查看副本集姿态使用 rs.status() 命令

# 副本集添加成员

添加副本集的成员，我们需要使用多条服务器来启动mongo服务。进入Mongo客户端，并使用rs.add()方法来添加副本集的成员。

## 语法

rs.add() 命令基本语法格式如下：

```
>rs.add(HOST_NAME:PORT)
```

## 实例

假设你已经启动了一个名为mongod1.net，端口号为27017的Mongo服务。 在客户端命令窗口使用rs.add() 命令将其添加到副本集中，命令如下所示：

```
>rs.add("mongod1.net:27017")
>
```

MongoDB中你只能通过主节点将Mongo服务添加到副本集中， 判断当前运行的Mongo服务是否为主节点可以使用命令db.isMaster() 。

MongoDB的副本集与我们常见的主从有所不同，主从在主机宕机后所有服务将停止，而副本集在主机宕机后，副本会接管主节点成为主节点，不会出现宕机的情况。

# MongoDB 分片

## 分片

在Mongodb里面存在另一种集群，就是分片技术,可以满足MongoDB数据量大量增长的需求。

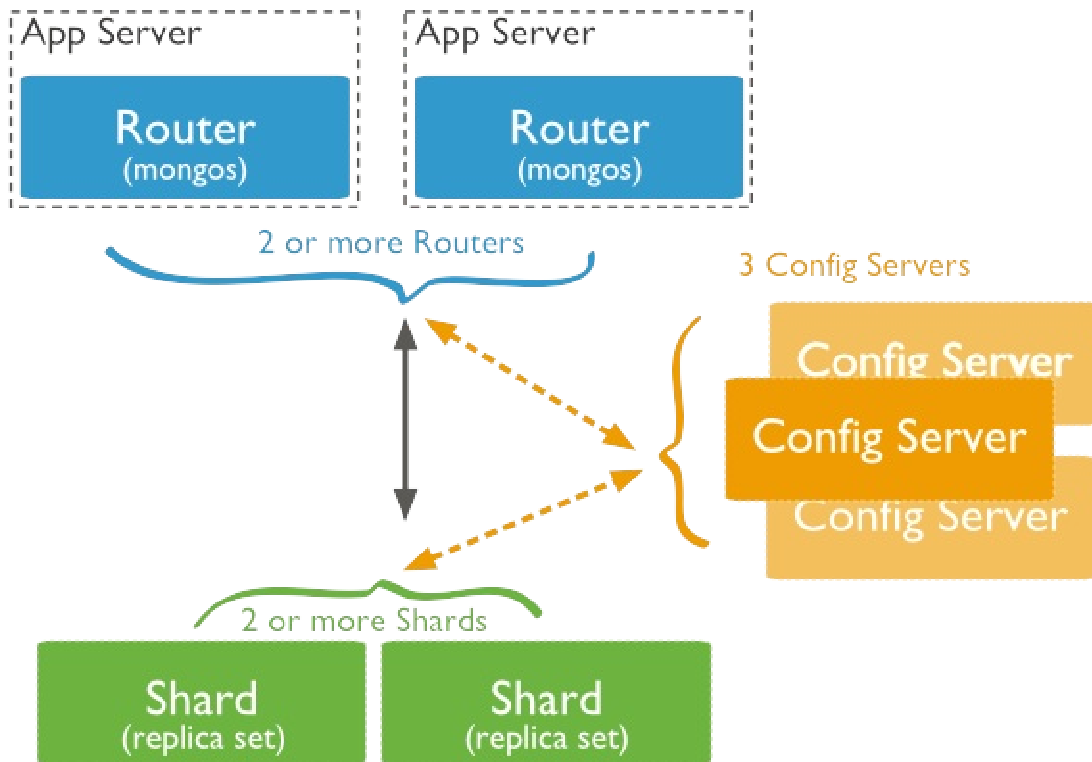
当MongoDB存储海量的数据时，一台机器可能不足以存储数据也足以提供可接受的读写吞吐量。这时，我们就可以通过在多台机器上分割数据，使得数据库系统能存储和处理更多的数据。

## 为什么使用分片

- 复制所有的写入操作到主节点
- 延迟的敏感数据会在主节点查询
- 单个副本集限制在12个节点
- 当请求量巨大时会出现内存不足。
- 本地磁盘不足
- 垂直扩展价格昂贵

## MongoDB分片

下图展示了在MongoDB中使用分片集群结构分布：




上图中主要有如下所述三个主要组件：

- **Shard:**

  用于存储实际的数据块，实际生产环境中一个shard server角色可由几台机器组个一个relica set承担，防止主机单点故障

- **Config Server:**

  mongod实例，存储了整个 ClusterMetadata，其中包括 chunk信息。

- **Query Routers:**

  前端路由，客户端由此接入，且让整个集群看上去像单一数据库，前端应用可以透明使用。

# 分片实例

分片结构端口分布如下：

```
Shard Server 1：27020
Shard Server 2：27021
Shard Server 3：27022
Shard Server 4：27023
Config Server ：27100
Route Process：40000
```

步骤一：启动Shard Server

```
[root@100 /]# mkdir -p /www/mongoDB/shard/s0
[root@100 /]# mkdir -p /www/mongoDB/shard/s1
[root@100 /]# mkdir -p /www/mongoDB/shard/s2
[root@100 /]# mkdir -p /www/mongoDB/shard/s3
[root@100 /]# mkdir -p /www/mongoDB/shard/log
[root@100 /]# /usr/local/mongoDB/bin/mongod --port 27020 --dbpath=/www/mongoDB/shard/s0 -
....
[root@100 /]# /usr/local/mongoDB/bin/mongod --port 27023 --dbpath=/www/mongoDB/shard/s3 -
```

步骤二： 启动Config Server

```
[root@100 /]# mkdir -p /www/mongoDB/shard/config
[root@100 /]# /usr/local/mongoDB/bin/mongod --port 27100 --dbpath=/www/mongoDB/shard/conf
```

注意：这里我们完全可以像启动普通mongodb服务一样启动，不需要添加—shardsvr和configsvr参数。因为这两个参数的作用就是改变启动端口的，所以我们自行指定了端口就可以。

步骤三： 启动Route Process

```
/usr/local/mongoDB/bin/mongos --port 40000 --configdb localhost:27100 --fork --logpath=/w
```

mongos启动参数中，chunkSize这一项是用来指定chunk的大小的，单位是MB，默认大小为200MB.

步骤四： 配置Sharding

接下来，我们使用MongoDB Shell登录到mongos，添加Shard节点

```
[root@100 shard]# /usr/local/mongoDB/bin/mongo admin --port 40000
MongoDB shell version: 2.0.7
connecting to: 127.0.0.1:40000/admin
mongos> db.runCommand({ addshard:"localhost:27020" })
{ "shardAdded" : "shard0000", "ok" : 1 }
......
mongos> db.runCommand({ addshard:"localhost:27029" })
{ "shardAdded" : "shard0009", "ok" : 1 }
mongos> db.runCommand({ enablesharding:"test" }) #设置分片存储的数据库
{ "ok" : 1 }
mongos> db.runCommand({ shardcollection: "test.log", key: { id:1,time:1}})
{ "collectionsharded" : "test.log", "ok" : 1 }
```

步骤五： 程序代码内无需太大更改，直接按照连接普通的mongo数据库那样，将数据库连接接入接口40000

# MongoDB 备份(mongodump)与恢复(mongorerstore)

## MongoDB数据备份

在Mongodb中我们使用mongodump命令来备份MongoDB数据。该命令可以导出所有数据到指定目录中。

mongodump命令可以通过参数指定导出的数据量级转存的服务器。

### 语法

mongodump命令脚本语法如下：

```
>mongodump -h dbhost -d dbname -o dbdirectory
```

- **-h**：

  MongDB所在服务器地址，例如：127.0.0.1，当然也可以指定端口号：127.0.0.1:27017

- **-d**：

  需要备份的数据库实例，例如：test

- **-o**：

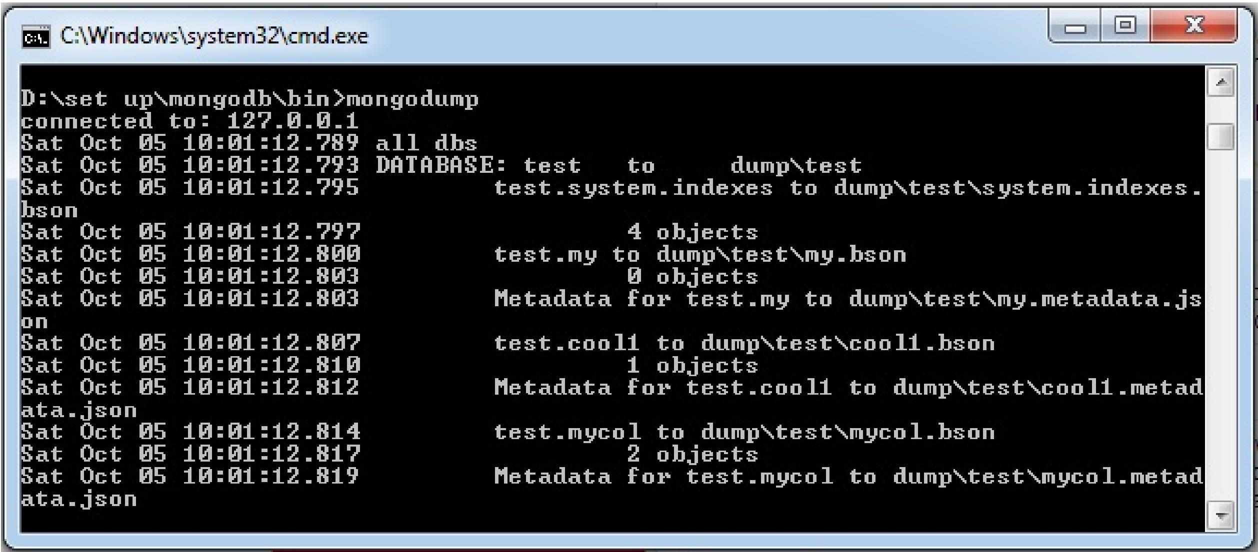
  备份的数据存放位置，例如：c:\data\dump，当然该目录需要提前建立，在备份完成后，系统自动在dump目录下建立一个test目录，这个目录里面存放该数据库实例的备份数据。

### 实例

在本地使用 27017 启动你的mongod服务。打开命令提示符窗口，进入MongoDB安装目录的bin目录输入命令mongodump:

```
>mongodump
```

执行以上命令后，客户端会连接到ip为 127.0.0.1 端口号为 27017 的MongoDB服务上，并备份所有数据到 bin/dump/ 目录中。命令输出结果如下：




mongodump 命令可选参数列表如下所示：

| 语法 | 描述 | 实例 |
| --- | --- | --- |
| mongodump --host HOST_NAME --port PORT_NUMBER | 该命令将备份所有MongoDB数据 | mongodump --host w3cschool.cc --port 27017 |
| mongodump --dbpath DB_PATH --out BACKUP_DIRECTORY | | mongodump --dbpath /data/db/ --out /data/backup/ |
| mongodump --collection COLLECTION --db DB_NAME | 该命令将备份指定数据库的集合。 | mongodump --collection mycol --db test |

# MongoDB数据恢复

mongodb使用 mongorerstore 命令来恢复备份的数据。

## 语法

mongorestore命令脚本语法如下：

```
>mongorestore -h dbhost -d dbname --directoryperdb dbdirectory
```

- **-h**：

  MongoDB所在服务器地址

- **-d**：

  需要恢复的数据库实例，例如：test，当然这个名称也可以和备份时候的不一样，比如test2

- **--directoryperdb**：

  备份数据所在位置，例如：c:\data\dump\test，这里为什么要多加一个test，而不是备份时候的dump，读者自己查看提示吧！

- **--drop**：

  恢复的时候，先删除当前数据，然后恢复备份的数据。就是说，恢复后，备份后添加修改的数据都会被删除，慎用哦！

接下来我们执行以下命令:

```
>mongorestore
```

执行以上命令输出结果如下：

```
C:\Windows\system32\cmd.exe
D:\set up\mongodb\bin>mongorestore
connected to: 127.0.0.1
Sat Oct 05 10:06:40.922 dump\test\cool1.bson
Sat Oct 05 10:06:40.924         going into namespace [test.cool1]
Sat Oct 05 10:06:40.933 warning: Restoring to test.cool1 without dropping. Resto
red data will be inserted without raising errors; check your server log
1 objects found
Sat Oct 05 10:06:41.003         Creating index: { key: { _id: 1 }, ns: "test.coo
l1", name: "_id_" }
Sat Oct 05 10:06:41.058 dump\test\my.bson
Sat Oct 05 10:06:41.058         going into namespace [test.my]
Sat Oct 05 10:06:41.062 warning: Restoring to test.my without dropping. Restored
 data will be inserted without raising errors; check your server log
Sat Oct 05 10:06:41.063 file dump\test\my.bson empty, skipping
Sat Oct 05 10:06:41.063         Creating index: { key: { _id: 1 }, ns: "test.my"
, name: "_id_" }
Sat Oct 05 10:06:41.066 dump\test\mycol.bson
Sat Oct 05 10:06:41.067         going into namespace [test.mycol]
Sat Oct 05 10:06:41.070 warning: Restoring to test.mycol without dropping. Resto
red data will be inserted without raising errors; check your server log
2 objects found
Sat Oct 05 10:06:41.077         Creating index: { key: { _id: 1 }, ns: "test.myc
ol", name: "_id_" }
Sat Oct 05 10:06:41.079         Creating index: { key: { name: 1 }, ns: "test.my
col", name: "name_1" }
```

# MongoDB 监控

在你已经安装部署并允许MongoDB服务后，你必须要了解MongoDB的运行情况，并查看MongoDB的性能。这样在大流量得情况下可以很好的应对并保证MongoDB正常运作。

MongoDB中提供了mongostat 和 mongotop 两个命令来监控MongoDB的运行情况。

## mongostat 命令

mongostat是mongodb自带的状态检测工具，在命令行下使用。它会间隔固定时间获取mongodb的当前运行状态，并输出。如果你发现数据库突然变慢或者有其他问题的话，你第一手的操作就考虑采用mongostat来查看mongo的状态。

启动你的Mongod服务，进入到你安装的MongoDB目录下的bin目录， 然后输入mongostat命令，如下所示：

```
D:\set up\mongodb\bin>mongostat
```

以上命令输出结果如下：

```
C:\Windows\system32\cmd.exe - mongostat
local:0.0%         0       0|0     0|0   115b     4k     2   19:20:24
insert  query update delete getmore command flushes mapped  vsize    res faults
 locked db idx miss %      qr|qw   ar|aw  netIn netOut  conn       time
    *0     *0     *0     *0       0     2|0       0  14.1g  28.3g    40m      0
local:0.0%         0       0|0     0|0   115b     4k     2   19:20:25
    *0     *0     *0     *0       0     2|0       0  14.1g  28.3g    40m      0
local:0.0%         0       0|0     0|0   115b     4k     2   19:20:26
    *0     *0     *0     *0       0     2|0       0  14.1g  28.3g    40m      0
local:0.0%         0       0|0     0|0   115b     4k     2   19:20:27
    *0     *0     *0     *0       0     2|0       0  14.1g  28.3g    40m      0
local:0.0%         0       0|0     0|0   115b     4k     2   19:20:28
    *0     *0     *0     *0       0     2|0       0  14.1g  28.3g    40m      0
local:0.0%         0       0|0     0|0   115b     4k     2   19:20:29
    *0     *0     *0     *0       0     2|0       0  14.1g  28.3g    40m      0
local:0.0%         0       0|0     0|0   115b     4k     2   19:20:30
    *0     *0     *0     *0       0     2|0       0  14.1g  28.3g    40m      0
local:0.0%         0       0|0     0|0   115b     4k     2   19:20:31
    *0     *0     *0     *0       0     2|0       0  14.1g  28.3g    40m      0
local:0.0%         0       0|0     0|0   115b     4k     2   19:20:32
    *0     *0     *0     *0       0     2|0       0  14.1g  28.3g    40m      0
local:0.0%         0       0|0     0|0   115b     4k     2   19:20:33
    *0     *0     *0     *0       0     2|0       0  14.1g  28.3g    40m      0
local:0.0%         0       0|0     0|0   115b     4k     2   19:20:34
insert  query update delete getmore command flushes mapped  vsize    res faults
 locked db idx miss %      qr|qw   ar|aw  netIn netOut  conn       time
    *0     *0     *0     *0       0     2|0       0  14.1g  28.3g    40m      0
local:0.0%         0       0|0     0|0   115b     4k     2   19:20:35
    *0     *0     *0     *0       0     2|0       0  14.1g  28.3g    40m      0
local:0.0%         0       0|0     0|0   115b     4k     2   19:20:36
    *0     *0     *0     *0       0     2|0       0  14.1g  28.3g    40m      0
local:0.0%         0       0|0     0|0   115b     4k     2   19:20:37
    *0     *0     *0     *0       0     2|0       0  14.1g  28.3g    40m      0
local:0.0%         0       0|0     0|0   115b     4k     2   19:20:38
    *0     *0     *0     *0       0     2|0       0  14.1g  28.3g    40m      0
local:0.0%         0       0|0     0|0   115b     4k     2   19:20:39
    *0     *0     *0     *0       0     2|0       0  14.1g  28.3g    40m      0
local:0.0%         0       0|0     0|0   115b     4k     2   19:20:40
    *0     *0     *0     *0       0     2|0       0  14.1g  28.3g    40m      0
local:0.0%         0       0|0     0|0   115b     4k     2   19:20:41
```

## mongotop 命令

mongotop也是mongodb下的一个内置工具，mongotop提供了一个方法，用来跟踪一个MongoDB的实例，查看哪些大量的时间花费在读取和写入数据。 mongotop提供每个集合的水平的统计数据。默认情况下，mongotop返回值的每一秒。

启动你的Mongod服务，进入到你安装的MongoDB目录下的bin目录， 然后输入mongotop命令，如下所示：

```
D:\set up\mongodb\bin>mongotop
```

以上命令执行输出结果如下：

```
C:\Windows\system32\cmd.exe - mongotop
           local.system.users          0ms          0ms          0ms
         local.system.replset          0ms          0ms          0ms
            local.startup_log          0ms          0ms          0ms

                           ns        total         read        write
2013-10-06T13:53:28
            test.system.users          0ms          0ms          0ms
           local.system.users          0ms          0ms          0ms
         local.system.replset          0ms          0ms          0ms
            local.startup_log          0ms          0ms          0ms

                           ns        total         read        write
2013-10-06T13:53:29
            test.system.users          0ms          0ms          0ms
           local.system.users          0ms          0ms          0ms
         local.system.replset          0ms          0ms          0ms
            local.startup_log          0ms          0ms          0ms

                           ns        total         read        write
2013-10-06T13:53:30
            test.system.users          0ms          0ms          0ms
           local.system.users          0ms          0ms          0ms
         local.system.replset          0ms          0ms          0ms
            local.startup_log          0ms          0ms          0ms
```

带参数实例

```
E:\mongodb-win32-x86_64-2.2.1\bin>mongotop 10
```

```
E:\mongodb-win32-x86_64-2.2.1\bin>mongotop
connected to: 127.0.0.1

                                ns       total        read       write
2013-03-29T04:12:17
              local.system.replset         0ms         0ms         0ms
           local.system.namespaces         0ms         0ms         0ms
              local.system.indexes         0ms         0ms         0ms
              admin.system.indexes         0ms         0ms         0ms
                            admin.         0ms         0ms         0ms
CpsCommodityInfo.system.namespaces         0ms         0ms         0ms
        CpsCommodityInfo.system.js         0ms         0ms         0ms

                                ns       total        read       write
2013-03-29T04:12:18
              local.system.replset         0ms         0ms         0ms
           local.system.namespaces         0ms         0ms         0ms
              local.system.indexes         0ms         0ms         0ms
              admin.system.indexes         0ms         0ms         0ms
                            admin.         0ms         0ms         0ms
CpsCommodityInfo.system.namespaces         0ms         0ms         0ms
        CpsCommodityInfo.system.js         0ms         0ms         0ms
```

后面的10是<*sleeptime*>参数，可以不使用，等待的时间长度，以秒为单位，mongotop等待调用之间。通过的默认mongotop返回数据的每一秒。

```
E:\mongodb-win32-x86_64-2.2.1\bin>mongotop --locks
```

报告每个数据库的锁的使用中，使用mongotop - 锁，这将产生以下输出：

```
E:\mongodb-win32-x86_64-2.2.1\bin>mongotop --locks
connected to: 127.0.0.1

                         db       total        read       write
2013-03-29T04:21:59
                      local         0ms         0ms         0ms
                      admin         0ms         0ms         0ms
           CpsCommodityInfo         0ms         0ms         0ms
                          .         0ms         0ms         0ms

                         db       total        read       write
2013-03-29T04:22:00
                      local         0ms         0ms         0ms
                      admin         0ms         0ms         0ms
           CpsCommodityInfo         0ms         0ms         0ms
                          .         0ms         0ms         0ms

                         db       total        read       write
2013-03-29T04:22:01
```

输出结果字段说明：

- **ns**：

  包含数据库命名空间，后者结合了数据库名称和集合。

- **db**：

  包含数据库的名称。名为 . 的数据库针对全局锁定，而非特定数据库。

- **total**：

  mongod花费的时间工作在这个命名空间提供总额。

- **read**：

  提供了大量的时间，这mongod花费在执行读操作，在此命名空间。

- **write**：

  提供这个命名空间进行写操作，这mongod花了大量的时间。

# MongoDB Java

环境配置

在Java程序中如果要使用MongoDB，你需要确保已经安装了Java环境及MongoDB JDBC 驱动。

你可以参考本站的Java教程来安装Java程序。现在让我们来检测你是否安装了 MongoDB JDBC 驱动。

- 首先你必须下载mongo jar包，下载地址：https://github.com/mongodb/mongo-java-driver/downloads, 请确保下载最新版本。
- 你需要将mongo.jar包含在你的 classpath 中。。

## 连接数据库

连接数据库，你需要指定数据库名称，如果指定的数据库不存在，mongo会自动创建数据库。

连接数据库的Java代码如下：

```
import com.mongodb.MongoClient;
import com.mongodb.MongoException;
import com.mongodb.WriteConcern;
import com.mongodb.DB;
import com.mongodb.DBCollection;
import com.mongodb.BasicDBObject;
import com.mongodb.DBObject;
import com.mongodb.DBCursor;
import com.mongodb.ServerAddress;
import java.util.Arrays;

public class MongoDBJDBC{
   public static void main( String args[] ){
      try{
         // 连接到 mongodb 服务
         MongoClient mongoClient = new MongoClient( "localhost" , 27017 );
         // 连接到数据库
         DB db = mongoClient.getDB( "test" );
         System.out.println("Connect to database successfully");
         boolean auth = db.authenticate(myUserName, myPassword);
         System.out.println("Authentication: "+auth);
      }catch(Exception e){
         System.err.println( e.getClass().getName() + ": " + e.getMessage() );
      }
   }
}
```

现在，让我们来编译运行程序并创建数据库test。

你可以更加你的实际环境改变MongoDB JDBC驱动的路径。

本实例将MongoDB JDBC启动包 mongo-2.10.1.jar 放在本地目录下:

```
$javac MongoDBJDBC.java
$java -classpath ".:mongo-2.10.1.jar" MongoDBJDBC
Connect to database successfully
Authentication: true
```

如果你使用的是Window系统，你可以按以下命令来编译执行程序：

```
$javac MongoDBJDBC.java
$java -classpath ".;mongo-2.10.1.jar" MongoDBJDBC
Connect to database successfully
Authentication: true
```

如果用户名及密码正确，则Authentication 的值为true。

# 创建集合

我们可以使用com.mongodb.DB类中的createCollection()来创建集合

代码片段如下：

```
import com.mongodb.MongoClient;
import com.mongodb.MongoException;
import com.mongodb.WriteConcern;
import com.mongodb.DB;
import com.mongodb.DBCollection;
import com.mongodb.BasicDBObject;
import com.mongodb.DBObject;
import com.mongodb.DBCursor;
import com.mongodb.ServerAddress;
import java.util.Arrays;

public class MongoDBJDBC{
   public static void main( String args[] ){
      try{
         // 连接到 mongodb 服务
         MongoClient mongoClient = new MongoClient( "localhost" , 27017 );
         // 连接到数据库
         DB db = mongoClient.getDB( "test" );
     System.out.println("Connect to database successfully");
         boolean auth = db.authenticate(myUserName, myPassword);
     System.out.println("Authentication: "+auth);
         DBCollection coll = db.createCollection("mycol");
         System.out.println("Collection created successfully");
      }catch(Exception e){
         System.err.println( e.getClass().getName() + ": " + e.getMessage() );
      }
   }
}
```

编译运行以上程序，输出结果如下:

```
Connect to database successfully
Authentication: true
Collection created successfully
```

# 获取集合

我们可以使用com.mongodb.DBCollection类的 getCollection() 方法来获取一个集合

代码片段如下：

```
import com.mongodb.MongoClient;
import com.mongodb.MongoException;
import com.mongodb.WriteConcern;
import com.mongodb.DB;
import com.mongodb.DBCollection;
import com.mongodb.BasicDBObject;
import com.mongodb.DBObject;
import com.mongodb.DBCursor;
import com.mongodb.ServerAddress;
import java.util.Arrays;

public class MongoDBJDBC{
   public static void main( String args[] ){
      try{
         // 连接到 mongodb 服务
         MongoClient mongoClient = new MongoClient( "localhost" , 27017 );
         // 连接到数据库
         DB db = mongoClient.getDB( "test" );
     System.out.println("Connect to database successfully");
         boolean auth = db.authenticate(myUserName, myPassword);
     System.out.println("Authentication: "+auth);
         DBCollection coll = db.createCollection("mycol");
         System.out.println("Collection created successfully");
         DBCollection coll = db.getCollection("mycol");
         System.out.println("Collection mycol selected successfully");
      }catch(Exception e){
         System.err.println( e.getClass().getName() + ": " + e.getMessage() );
      }
   }
}
```

编译运行以上程序，输出结果如下:

```
Connect to database successfully
Authentication: true
Collection created successfully
Collection mycol selected successfully
```

# 插入文档

我们可以使用com.mongodb.DBCollection类的 insert() 方法来插入一个文档

代码片段如下：

```
import com.mongodb.MongoClient;
import com.mongodb.MongoException;
import com.mongodb.WriteConcern;
import com.mongodb.DB;
import com.mongodb.DBCollection;
import com.mongodb.BasicDBObject;
import com.mongodb.DBObject;
import com.mongodb.DBCursor;
import com.mongodb.ServerAddress;
import java.util.Arrays;

public class MongoDBJDBC{
   public static void main( String args[] ){
      try{
         // 连接到 mongodb 服务
         MongoClient mongoClient = new MongoClient( "localhost" , 27017 );
         // 连接到数据库
         DB db = mongoClient.getDB( "test" );
     System.out.println("Connect to database successfully");
         boolean auth = db.authenticate(myUserName, myPassword);
     System.out.println("Authentication: "+auth);
         DBCollection coll = db.getCollection("mycol");
         System.out.println("Collection mycol selected successfully");
         BasicDBObject doc = new BasicDBObject("title", "MongoDB").
            append("description", "database").
            append("likes", 100).
            append("url", "http://www.w3cschool.cc/mongodb/").
            append("by", "w3cschool.cc");
         coll.insert(doc);
         System.out.println("Document inserted successfully");
      }catch(Exception e){
         System.err.println( e.getClass().getName() + ": " + e.getMessage() );
      }
   }
}
```

编译运行以上程序，输出结果如下:

```
Connect to database successfully
Authentication: true
Collection mycol selected successfully
Document inserted successfully
```

# 检索所有文档

我们可以使用com.mongodb.DBCollection类中的 find() 方法来获取集合中的所有文档。

此方法返回一个游标，所以你需要遍历这个游标。

代码片段如下：

```
import com.mongodb.MongoClient;
import com.mongodb.MongoException;
import com.mongodb.WriteConcern;
import com.mongodb.DB;
import com.mongodb.DBCollection;
import com.mongodb.BasicDBObject;
import com.mongodb.DBObject;
import com.mongodb.DBCursor;
import com.mongodb.ServerAddress;
import java.util.Arrays;

public class MongoDBJDBC{
   public static void main( String args[] ){
      try{
        // 连接到 mongodb 服务
         MongoClient mongoClient = new MongoClient( "localhost" , 27017 );
          // 连接到数据库
         DB db = mongoClient.getDB( "test" );
     System.out.println("Connect to database successfully");
         boolean auth = db.authenticate(myUserName, myPassword);
     System.out.println("Authentication: "+auth);
         DBCollection coll = db.getCollection("mycol");
         System.out.println("Collection mycol selected successfully");
         DBCursor cursor = coll.find();
         int i=1;
         while (cursor.hasNext()) {
            System.out.println("Inserted Document: "+i);
            System.out.println(cursor.next());
            i++;
         }
      }catch(Exception e){
         System.err.println( e.getClass().getName() + ": " + e.getMessage() );
      }
   }
}
```

编译运行以上程序，输出结果如下:

```
Connect to database successfully
Authentication: true
Collection mycol selected successfully
Inserted Document: 1
{
   "_id" : ObjectId(7df78ad8902c),
   "title": "MongoDB",
   "description": "database",
   "likes": 100,
   "url": "http://www.w3cschool.cc/mongodb/",
   "by": "w3cschool.cc"
}
```

# 更新文档

你可以使用 com.mongodb.DBCollection 类中的 update() 方法来更新集合中的文档。

代码片段如下：

```
import com.mongodb.MongoClient;
import com.mongodb.MongoException;
import com.mongodb.WriteConcern;
import com.mongodb.DB;
import com.mongodb.DBCollection;
import com.mongodb.BasicDBObject;
import com.mongodb.DBObject;
import com.mongodb.DBCursor;
import com.mongodb.ServerAddress;
import java.util.Arrays;

public class MongoDBJDBC{
   public static void main( String args[] ){
      try{
     // 连接到Mongodb服务
         MongoClient mongoClient = new MongoClient( "localhost" , 27017 );
         // 连接到你的数据库
         DB db = mongoClient.getDB( "test" );
     System.out.println("Connect to database successfully");
         boolean auth = db.authenticate(myUserName, myPassword);
     System.out.println("Authentication: "+auth);
         DBCollection coll = db.getCollection("mycol");
         System.out.println("Collection mycol selected successfully");
         DBCursor cursor = coll.find();
         while (cursor.hasNext()) {
            DBObject updateDocument = cursor.next();
            updateDocument.put("likes","200")
            col1.update(updateDocument);
         }
         System.out.println("Document updated successfully");
         cursor = coll.find();
         int i=1;
         while (cursor.hasNext()) {
            System.out.println("Updated Document: "+i);
            System.out.println(cursor.next());
            i++;
         }
      }catch(Exception e){
         System.err.println( e.getClass().getName() + ": " + e.getMessage() );
      }
   }
}
```

编译运行以上程序，输出结果如下:

```
Connect to database successfully
Authentication: true
Collection mycol selected successfully
Document updated successfully
Updated Document: 1
{
   "_id" : ObjectId(7df78ad8902c),
   "title": "MongoDB",
   "description": "database",
   "likes": 100,
   "url": "http://www.w3cschool.cc/mongodb/",
   "by": "w3cschool.cc"
}
```

# 删除第一个文档

要删除集合中的第一个文档，首先你需要使用com.mongodb.DBCollection类中的 findOne()方法来获取第一个文档，然后使用remove 方法删除。

代码片段如下：

```
 import com.mongodb.MongoClient;
import com.mongodb.MongoException;
import com.mongodb.WriteConcern;
import com.mongodb.DB;
import com.mongodb.DBCollection;
import com.mongodb.BasicDBObject;
import com.mongodb.DBObject;
import com.mongodb.DBCursor;
import com.mongodb.ServerAddress;
import java.util.Arrays;

public class MongoDBJDBC{
   public static void main( String args[] ){
      try{
     // 连接到Mongodb服务
         MongoClient mongoClient = new MongoClient( "localhost" , 27017 );
         // 连接到你的数据库
         DB db = mongoClient.getDB( "test" );
     System.out.println("Connect to database successfully");
         boolean auth = db.authenticate(myUserName, myPassword);
     System.out.println("Authentication: "+auth);
         DBCollection coll = db.getCollection("mycol");
         System.out.println("Collection mycol selected successfully");
         DBObject myDoc = coll.findOne();
         col1.remove(myDoc);
         DBCursor cursor = coll.find();
         int i=1;
         while (cursor.hasNext()) {
            System.out.println("Inserted Document: "+i);
            System.out.println(cursor.next());
            i++;
         }
         System.out.println("Document deleted successfully");
      }catch(Exception e){
         System.err.println( e.getClass().getName() + ": " + e.getMessage() );
      }
   }
}
```

编译运行以上程序，输出结果如下:

```
Connect to database successfully
Authentication: true
Collection mycol selected successfully
Document deleted successfully
```

你还可以使用 save(), limit(), skip(), sort() 等方法来操作MongoDB数据库。

# MongoDB PHP

在php中使用mongodb你必须使用 mongodb的php驱动。

MongoDB PHP在各平台上的安装及驱动包下载请查看:PHP安装MongoDB扩展驱动

## 确保连接及选择一个数据库

为了确保正确连接，你需要指定数据库名，如果数据库在mongoDB中不存在，mongoDB会自动创建

代码片段如下：

```
<?php
   // 连接到mongodb
   $m = new MongoClient();
   echo "Connection to database successfully";
   // 选择一个数据库
   $db = $m->mydb;
   echo "Database mydb selected";
?>
```

执行以上程序，输出结果如下：

```
Connection to database successfully
Database mydb selected
```

## 创建集合

创建集合的代码片段如下：

```
<?php
   // 连接到mongodb
   $m = new MongoClient();
   echo "Connection to database successfully";
   // 选择一个数据库
   $db = $m->mydb;
   echo "Database mydb selected";
   $collection = $db->createCollection("mycol");
   echo "Collection created succsessfully";
?>
```

执行以上程序，输出结果如下：

```
Connection to database successfully
Database mydb selected
Collection created succsessfully
```

# 插入文档

在mongoDB中使用 insert() 方法插入文档：

插入文档代码片段如下：

```
<?php
   // 连接到mongodb
   $m = new MongoClient();
   echo "Connection to database successfully";
   // 选择一个数据库
   $db = $m->mydb;
   echo "Database mydb selected";
   $collection = $db->mycol;
   echo "Collection selected succsessfully";
   $document = array(
      "title" => "MongoDB",
      "description" => "database",
      "likes" => 100,
      "url" => "http://www.w3cschool.cc/mongodb/",
      "by", "w3cschool.cc"
   );
   $collection->insert($document);
   echo "Document inserted successfully";
?>
```

执行以上程序，输出结果如下：

```
Connection to database successfully
Database mydb selected
Collection selected succsessfully
Document inserted successfully
```

# 查找文档

使用find() 方法来读取集合中的文档。

读取使用文档的代码片段如下：

```
<?php
   // 连接到mongodb
   $m = new MongoClient();
   echo "Connection to database successfully";
   // 选择一个数据库
   $db = $m->mydb;
   echo "Database mydb selected";
   $collection = $db->mycol;
   echo "Collection selected succsessfully";

   $cursor = $collection->find();
   // 迭代显示文档标题
   foreach ($cursor as $document) {
      echo $document["title"] . "\n";
   }
?>
```

执行以上程序，输出结果如下：

```
Connection to database successfully
Database mydb selected
Collection selected succsessfully
{
   "title": "MongoDB"
}
```

# 更新文档

使用 update() 方法来更新文档。

以下实例将更新文档中的标题为' MongoDB Tutorial'， 代码片段如下：

```
<pre>
<?php
   // 连接到mongodb
   $m = new MongoClient();
   echo "Connection to database successfully";
   // 选择一个数据库
   $db = $m->mydb;
   echo "Database mydb selected";
   $collection = $db->mycol;
   echo "Collection selected succsessfully";

   // 更新文档
   $collection->update(array("title"=>"MongoDB"), array('$set'=>array("title"=>"MongoDB T
   echo "Document updated successfully";
   // 显示更新后的文档
   $cursor = $collection->find();
   // 循环显示文档标题
   echo "Updated document";
   foreach ($cursor as $document) {
      echo $document["title"] . "\n";
   }
?>
```

执行以上程序，输出结果如下：

```
Connection to database successfully
Database mydb selected
Collection selected succsessfully
Document updated successfully
Updated document
{
   "title": "MongoDB Tutorial"
}
```

# 删除文档

使用 remove() 方法来删除文档。

以下实例中我们将移除 'title' 为 'MongoDB Tutorial' 的数据记录。， 代码片段如下：

```
<?php
   // 连接到mongodb
   $m = new MongoClient();
   echo "Connection to database successfully";
   // 选择一个数据库
   $db = $m->mydb;
   echo "Database mydb selected";
   $collection = $db->mycol;
   echo "Collection selected succsessfully";

   // 移除文档
   $collection->remove(array("title"=>"MongoDB Tutorial"),false);
   echo "Documents deleted successfully";

   // 显示可用文档数据
   $cursor = $collection->find();
   // iterate cursor to display title of documents
   echo "Updated document";
   foreach ($cursor as $document) {
      echo $document["title"] . "\n";
   }
?>
```

执行以上程序，输出结果如下：

```
Connection to database successfully
Database mydb selected
Collection selected succsessfully
Documents deleted successfully
```

除了以上实例外，在php中你还可以使用findOne(), save(), limit(), skip(), sort()等方法来操作Mongodb数据库。

# MongoDB 关系

MongoDB 的关系表示多个文档之间在逻辑上的相互联系。

文档间可以通过嵌入和引用来建立联系。

MongoDB 中的关系可以是：

- 1:1 (1对1)
- 1: N (1对多)
- N: 1 (多对1)
- N: N (多对多)

接下来我们来考虑下用户与用户地址的关系。

一个用户可以有多个地址，所以是一对多的关系。

以下是 **user** 文档的简单结构：

```
{
   "_id":ObjectId("52ffc33cd85242f436000001"),
   "name": "Tom Hanks",
   "contact": "987654321",
   "dob": "01-01-1991"
}
```

以下是 **address** 文档的简单结构：

```
{
   "_id":ObjectId("52ffc4a5d85242602e000000"),
   "building": "22 A, Indiana Apt",
   "pincode": 123456,
   "city": "Los Angeles",
   "state": "California"
}
```

## 嵌入式关系

使用嵌入式方法，我们可以把用户地址嵌入到用户的文档中：

```
   "_id":ObjectId("52ffc33cd85242f436000001"),
   "contact": "987654321",
   "dob": "01-01-1991",
   "name": "Tom Benzamin",
   "address": [
      {
         "building": "22 A, Indiana Apt",
         "pincode": 123456,
         "city": "Los Angeles",
         "state": "California"
      },
      {
         "building": "170 A, Acropolis Apt",
         "pincode": 456789,
         "city": "Chicago",
         "state": "Illinois"
      }]
}
```

以上数据保存在单一的文档中，可以比较容易的获取很维护数据。 你可以这样查询用户的地址：

```
>db.users.findOne({"name":"Tom Benzamin"},{"address":1})
```

注意：以上查询中 **db** 和 **users** 表示数据库和集合。

这种数据结构的缺点是，如果用户和用户地址在不断增加，数据量不断变大，会影响读写性能。

# 引用式关系

引用式关系是设计数据库时经常用到的方法，这种方法把用户数据文档和用户地址数据文档分开，通过引用文档的 **id** 字段来建立关系。

```
{
   "_id":ObjectId("52ffc33cd85242f436000001"),
   "contact": "987654321",
   "dob": "01-01-1991",
   "name": "Tom Benzamin",
   "address_ids": [
      ObjectId("52ffc4a5d85242602e000000"),
      ObjectId("52ffc4a5d85242602e000001")
   ]
}
```

以上实例中，用户文档的 **address_ids** 字段包含用户地址的对象id（ObjectId）数组。

我们可以读取这些用户地址的对象id（ObjectId）来获取用户的详细地址信息。

这种方法需要两次查询，第一次查询用户地址的对象id（ObjectId），第二次通过查询的id获取用户的详细地址信息。

```
>var result = db.users.findOne({"name":"Tom Benzamin"},{"address_ids":1})
>var addresses = db.address.find({"_id":{"$in":result["address_ids"]}})
```

# MongoDB 数据库引用

在上一章节MongoDB关系中我们提到了MongoDB的引用来规范数据结构文档。

MongoDB 引用有两种：

- 手动引用（Manual References）
- DBRefs

## DBRefs vs 手动引用

考虑这样的一个场景，我们在不同的集合中 (address_home, address_office, address_mailing, 等)存储不同的地址（住址，办公室地址，邮件地址等）。

这样，我们在调用不同地址时，也需要指定集合，一个文档从多个集合引用文档，我们应该使用 DBRefs。

## 使用 DBRefs

DBRef的形式：

```
{ $ref : , $id : , $db :  }
```

三个字段表示的意义为：

- $ref：集合名称
- $id：引用的id
- $db:数据库名称，可选参数

以下实例中用户数据文档使用了 DBRef, 字段 address：

```
{
   "_id":ObjectId("53402597d852426020000002"),
   "address": {
   "$ref": "address_home",
   "$id": ObjectId("534009e4d852427820000002"),
   "$db": "w3cschoolcc"},
   "contact": "987654321",
   "dob": "01-01-1991",
   "name": "Tom Benzamin"
}
```

**address** DBRef 字段指定了引用的地址文档是在 address_home 集合下的 w3cschoolcc 数据库，id 为 534009e4d852427820000002。

以下代码中，我们通过指定 $ref 参数（address_home 集合）来查找集合中指定id的用户地址信息：

```
>var user = db.users.findOne({"name":"Tom Benzamin"})
>var dbRef = user.address
>db[dbRef.$ref].findOne({"_id":(dbRef.$id)})
```

以上实例返回了 address_home 集合中的地址数据：

```
{
   "_id" : ObjectId("534009e4d852427820000002"),
   "building" : "22 A, Indiana Apt",
   "pincode" : 123456,
   "city" : "Los Angeles",
   "state" : "California"
}
```

# MongoDB 覆盖索引查询

官方的MongoDB的文档中说明，覆盖查询是以下的查询：

- 所有的查询字段是索引的一部分
- 所有的查询返回字段在同一个索引中

由于所有出现在查询中的字段是索引的一部分， MongoDB 无需在整个数据文档中检索匹配查询条件和返回使用相同索引的查询结果。

因为索引存在于RAM中，从索引中获取数据比通过扫描文档读取数据要快得多。

## 使用覆盖索引查询

为了测试盖索引查询，使用以下 users 集合:

```
{
   "_id": ObjectId("53402597d852426020000002"),
   "contact": "987654321",
   "dob": "01-01-1991",
   "gender": "M",
   "name": "Tom Benzamin",
   "user_name": "tombenzamin"
}
```

我们在 users 集合中创建联合索引，字段为 gender 和 user_name :

```
>db.users.ensureIndex({gender:1,user_name:1})
```

现在，该索引会覆盖以下查询：

```
>db.users.find({gender:"M"},{user_name:1,_id:0})
```

也就是说，对于上述查询，MongoDB的不会去数据库文件中查找。相反，它会从索引中提取数据，这是非常快速的数据查询。

由于我们的索引中不包括 _id 字段，_id在查询中会默认返回，我们可以在MongoDB的查询结果集中排除它。

下面的实例没有排除_id，查询就不会被覆盖：

```
>db.users.find({gender:"M"},{user_name:1})
```

最后，如果是以下的查询，不能使用覆盖索引查询：

- 所有索引字段是一个数组

# MongoDB 查询分析

MongoDB 查询分析可以确保我们建议的索引是否有效，是查询语句性能分析的重要工具。

MongoDB 查询分析常用函数有：explain() 和 hint()。

# 使用 explain()

explain 操作提供了查询信息，使用索引及查询统计等。有利于我们对索引的优化。

接下来我们在 users 集合中创建 gender 和 user_name 的索引：

```
>db.users.ensureIndex({gender:1,user_name:1})
</p>
<p>现在在查询语句中使用 explain ：</p>
<pre>
>db.users.find({gender:"M"},{user_name:1,_id:0}).explain()
```

以上的 explain() 查询返回如下结果：

```
{
   "cursor" : "BtreeCursor gender_1_user_name_1",
   "isMultiKey" : false,
   "n" : 1,
   "nscannedObjects" : 0,
   "nscanned" : 1,
   "nscannedObjectsAllPlans" : 0,
   "nscannedAllPlans" : 1,
   "scanAndOrder" : false,
   "indexOnly" : true,
   "nYields" : 0,
   "nChunkSkips" : 0,
   "millis" : 0,
   "indexBounds" : {
      "gender" : [
         [
            "M",
            "M"
         ]
      ],
      "user_name" : [
         [
            {
               "$minElement" : 1
            },
            {
               "$maxElement" : 1
            }
         ]
      ]
   }
}
```

现在，我们看看这个结果集的字段：

- **indexOnly**: 字段为 true ，表示我们使用了索引。
- **cursor**：因为这个查询使用了索引，MongoDB中索引存储在B树结构中，所以这是也使用了BtreeCursor类型的游标。如果没有使用索引，游标的类型是BasicCursor。这个键还会给出你所使用的索引的名称，你通过这个名称可以查看当前数据库下的system.indexes集合（系统自动创建，由于存储索引信息，这个稍微会提到）来得到索引的详细信息。
- **n**：当前查询返回的文档数量。
- **nscanned/nscannedObjects**：表明当前这次查询一共扫描了集合中多少个文档，我们的目的是，让这个数值和返回文档的数量越接近越好。
- **millis**：当前查询所需时间，毫秒数。
- **indexBounds**：当前查询具体使用的索引。

# 使用 hint()

虽然MongoDB查询优化器一般工作的很不错，但是也可以使用hints来强迫MongoDB使用一个指定的索引。

这种方法某些情形下会提升性能。 一个有索引的collection并且执行一个多字段的查询(一些字段已经索引了)。

如下查询实例指定了使用 gender 和 user_name 索引字段来查询：

```
>db.users.find({gender:"M"},{user_name:1,_id:0}).hint({gender:1,user_name:1})
```

可以使用 explain() 函数来分析以上查询：

```
>db.users.find({gender:"M"},{user_name:1,_id:0}).hint({gender:1,user_name:1}).explain()
```

# MongoDB 原子操作

mongodb不支持事务，所以，在你的项目中应用时，要注意这点。无论什么设计，都不要要求mongodb保证数据的完整性。

但是mongodb提供了许多原子操作，比如文档的保存，修改，删除等，都是原子操作。

所谓原子操作就是要么这个文档保存到Mongodb，要么没有保存到Mongodb，不会出现查询到的文档没有保存完整的情况。

## 原子操作数据模型

考虑下面的例子，图书馆的书籍及结账信息。

实例说明了在一个相同的文档中如何确保嵌入字段关联原子操作（update：更新）的字段是同步的。

```
book = {
          _id: 123456789,
          title: "MongoDB: The Definitive Guide",
          author: [ "Kristina Chodorow", "Mike Dirolf" ],
          published_date: ISODate("2010-09-24"),
          pages: 216,
          language: "English",
          publisher_id: "oreilly",
          available: 3,
          checkout: [ { by: "joe", date: ISODate("2012-10-15") } ]
        }
```

你可以使用 db.collection.findAndModify() 方法来判断书籍是否可结算并更新新的结算信息。

在同一个文档中嵌入的 available 和 checkout 字段来确保这些字段是同步更新的:

```
db.books.findAndModify ( {
   query: {
            _id: 123456789,
            available: { $gt: 0 }
          },
   update: {
             $inc: { available: -1 },
             $push: { checkout: { by: "abc", date: new Date() } }
           }
} )
```

## 原子操作常用命令

### $set

用来指定一个键并更新键值，若键不存在并创建。

```
{ $set : { field : value } }
```

### $unset

用来删除一个键。

```
{ $unset : { field : 1} }
```

### $inc

$inc可以对文档的某个值为数字型（只能为满足要求的数字）的键进行增减的操作。

```
{ $inc : { field : value } }
```

### $push

用法：

```
{ $push : { field : value } }
```

把value追加到field里面去，field一定要是数组类型才行，如果field不存在，会新增一个数组类型加进去。

### $pushAll

同$push,只是一次可以追加多个值到一个数组字段内。

```
{ $pushAll : { field : value_array } }
```

### $pull

从数组field内删除一个等于value值。

```
{ $pull : { field : _value } }
```

### $addToSet

增加一个值到数组内，而且只有当这个值不在数组内才增加。

### $pop

删除数组的第一个或最后一个元素

```
{ $pop : { field : 1 } }
```

### $rename

修改字段名称

```
{ $rename : { old_field_name : new_field_name } }
```

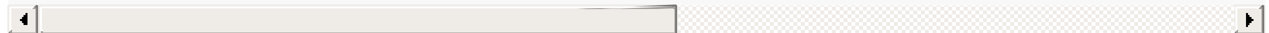
### $bit

位操作，integer类型

```
{$bit : { field : {and : 5}}}
```

### 偏移操作符

```
> t.find() { "_id" : ObjectId("4b97e62bf1d8c7152c9ccb74"), "title" : "ABC", "comments" :
> t.update( {'comments.by':'joe'}, {$inc:{'comments.$.votes':1}}, false, true )
> t.find() { "_id" : ObjectId("4b97e62bf1d8c7152c9ccb74"), "title" : "ABC", "comments" :
```



# MongoDB 高级索引

考虑以下文档集合（users ）：

```
{
   "address": {
      "city": "Los Angeles",
      "state": "California",
      "pincode": "123"
   },
   "tags": [
      "music",
      "cricket",
      "blogs"
   ],
   "name": "Tom Benzamin"
}
```

以上文档包含了 address 子文档和 tags 数组。

## 索引数组字段

假设我们基于标签来检索用户，为此我们需要对集合中的数组 tags 建立索引。

在数组中创建索引，需要对数组中的每个字段依次建立索引。所以在我们为数组 tags 创建索引时，会为 music、cricket、blogs三个值建立单独的索引。

使用以下命令创建数组索引：

```
>db.users.ensureIndex({"tags":1})
```

创建索引后，我们可以这样检索集合的 tags 字段：

```
>db.users.find({tags:"cricket"})
```

为了验证我们使用使用了索引，可以使用 explain 命令：

```
>db.users.find({tags:"cricket"}).explain()
```

以上命令执行结果中会显示 "cursor" : "BtreeCursor tags_1" ，则表示已经使用了索引。

## 索引子文档字段

假设我们需要通过city、state、pincode字段来检索文档，由于这些字段是子文档的字段，所以我们需要对子文档建立索引。

为子文档的三个字段创建索引，命令如下：

```
>db.users.ensureIndex({"address.city":1,"address.state":1,"address.pincode":1})
```

一旦创建索引，我们可以使用子文档的字段来检索数据：

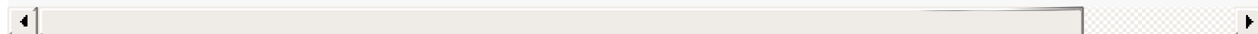
```
>db.users.find({"address.city":"Los Angeles"})
```

记住查询表达式必须遵循指定的索引的顺序。所以上面创建的索引将支持以下查询：

```
>db.users.find({"address.city":"Los Angeles","address.state":"California"})
```

同样支持以下查询：

```
>db.users.find({"address.city":"LosAngeles","address.state":"California","address.pincode
```



# MongoDB 索引限制

## 额外开销

每个索引占据一定的存储空间，在进行插入，更新和删除操作时也需要对索引进行操作。所以，如果你很少对集合进行读取操作，建议不使用索引。

## 内存(RAM)使用

由于索引是存储在内存(RAM)中,你应该确保该索引的大小不超过内存的限制。

如果索引的大小大于内存的限制，MongoDB会删除一些索引，这将导致性能下降。

## 查询限制

索引不能被以下的查询使用：

- 正则表达式及非操作符，如 $nin, $not, 等。
- 算术运算符，如 $mod, 等。
- $where 子句

所以，检测你的语句是否使用索引是一个好的习惯，可以用explain来查看。

## 索引键限制

从2.6版本开始，如果现有的索引字段的值超过索引键的限制，MongoDB中不会创建索引。

## 插入文档超过索引键限制

如果文档的索引字段值超过了索引键的限制，MongoDB不会将任何文档转换成索引的集合。与mongorestore和mongoimport工具类似。

## 最大范围

- 集合中索引不能超过64个
- 索引名的长度不能超过125个字符
- 一个复合索引最多可以有31个字段

# MongoDB ObjectId

在前面几个章节中我们已经使用了MongoDB 的对象 Id(ObjectId)。

在本章节中，我们将了解的ObjectId的结构。

ObjectId 是一个12字节 BSON 类型数据，有以下格式：

- 前4个字节表示时间戳
- 接下来的3个字节是机器标识码
- 紧接的两个字节由进程id组成（PID）
- 最后三个字节是随机数。

MongoDB中存储的文档必须有一个"_id"键。这个键的值可以是任何类型的，默认是个ObjectId对象。

在一个集合里面，每个集合都有唯一的"_id"值，来确保集合里面每个文档都能被唯一标识。

MongoDB采用ObjectId，而不是其他比较常规的做法（比如自动增加的主键）的主要原因，因为在多个 服务器上同步自动增加主键值既费力还费时。

## 创建信的ObjectId

使用以下代码生成新的ObjectId：

```
>newObjectId = ObjectId()
```

上面的语句返回以下唯一生成的id：

```
ObjectId("5349b4ddd2781d08c09890f3")
```

你也可以使用生成的id来取代MongoDB自动生成的ObjectId：

```
>myObjectId = ObjectId("5349b4ddd2781d08c09890f4")
```

## 创建文档的时间戳

由于 ObjectId 中存储了 4 个字节的时间戳，所以你不需要为你的文档保存时间戳字段，你可以通过 getTimestamp 函数来获取文档的创建时间:

```
>ObjectId("5349b4ddd2781d08c09890f4").getTimestamp()
```

以上代码将返回 ISO 格式的文档创建时间：

```
ISODate("2014-04-12T21:49:17Z")
```

# ObjectId 转换为字符串

在某些情况下，您可能需要将ObjectId转换为字符串格式。你可以使用下面的代码：

```
>new ObjectId.str
```

以上代码将返回Guid格式的字符串：：

```
5349b4ddd2781d08c09890f3
```

# MongoDB Map Reduce

Map-Reduce是一种计算模型，简单的说就是将大批量的工作（数据）分解（MAP）执行，然后再将结果合并成最终结果（REDUCE）。

MongoDB提供的Map-Reduce非常灵活，对于大规模数据分析也相当实用。

# MapReduce 命令

以下是MapReduce的基本语法：

```
>db.collection.mapReduce(
   function() {emit(key,value);},  //map 函数
   function(key,values) {return reduceFunction},   //reduce 函数
   {
      out: collection,
      query: document,
      sort: document,
      limit: number
   }
)
```

使用 MapReduce 要实现两个函数 Map 函数和 Reduce 函数,Map 函数调用 emit(key, value), 遍历 collection 中所有的记录, 将key 与 value 传递给 Reduce 函数进行处理。

Map 函数必须调用 emit(key, value) 返回键值对。

参数说明:

- **map** ：映射函数 (生成键值对序列,作为 reduce 函数参数)。
- **reduce** 统计函数，reduce函数的任务就是将key-values变成key-value，也就是把values数组变成一个单一的值value。。
- **out** 统计结果存放集合 (不指定则使用临时集合,在客户端断开后自动删除)。
- **query** 一个筛选条件，只有满足条件的文档才会调用map函数。（query。limit，sort可以随意组合）
- **sort** 和limit结合的sort排序参数（也是在发往map函数前给文档排序），可以优化分组机制
- **limit** 发往map函数的文档数量的上限（要是没有limit，单独使用sort的用处不大）

# 使用 MapReduce

考虑以下文档结构存储用户的文章，文档存储了用户的 user_name 和文章的 status字段：

```
{
   "post_text": "w3cschool.cc 菜鸟教程，最全的技术文档。",
   "user_name": "mark",
   "status":"active"
}
```

现在，我们将在 posts 集合中使用 mapReduce 函数来选取已发布的文章，并通过user_name 分组，计算每个用户的文章数：

```
>db.posts.mapReduce(
   function() { emit(this.user_id,1); },
   function(key, values) {return Array.sum(values)},
      {
         query:{status:"active"},
         out:"post_total"
      }
)
```

以上 mapReduce 输出结果为：

```
{
   "result" : "post_total",
   "timeMillis" : 9,
   "counts" : {
      "input" : 4,
      "emit" : 4,
      "reduce" : 2,
      "output" : 2
   },
   "ok" : 1,
}
```

结果表明，共有4个符合查询条件（status:"active"）的文档， 在map函数中生成了4个键值对文档，最后使用reduce函数将相同的键值分为两组。

具体参数说明：

- result：储存结果的collection的名字,这是个临时集合，MapReduce的连接关闭后自动就被删除了。
- timeMillis：执行花费的时间，毫秒为单位
- input：满足条件被发送到map函数的文档个数
- emit：在map函数中emit被调用的次数，也就是所有集合中的数据总量
- ouput：结果集合中的文档个数（**count**对调试非常有帮助）
- ok：是否成功，成功为1
- err：如果失败，这里可以有失败原因，不过从经验上来看，原因比较模糊，作用不大

使用 find 操作符来查看 mapReduce 的查询结果：

```
>db.posts.mapReduce(
   function() { emit(this.user_id,1); },
   function(key, values) {return Array.sum(values)},
      {
         query:{status:"active"},
         out:"post_total"
      }
).find()
```

以上查询显示如下结果，两个用户 tom 和 mark 有两个发布的文章:

```
{ "_id" : "tom", "value" : 2 }
{ "_id" : "mark", "value" : 2 }
```

用类似的方式，MapReduce可以被用来构建大型复杂的聚合查询。

Map函数和Reduce函数可以使用 JavaScript 来实现，是的MapReduce的使用非常灵活和强大。

# MongoDB 全文检索

全文检索对每一个词建立一个索引，指明该词在文章中出现的次数和位置，当用户查询时，检索程序就根据事先建立的索引进行查找，并将查找的结果反馈给用户的检索方式。

这个过程类似于通过字典中的检索字表查字的过程。

MongoDB 从 2.4 版本开始支持全文检索，目前支持15种语言(暂时不支持中文)的全文索引。

- danish
- dutch
- english
- finnish
- french
- german
- hungarian
- italian
- norwegian
- portuguese
- romanian
- russian
- spanish
- swedish
- turkish

## 启用全文检索

MongoDB 在 2.6 版本以后是默认开启全文检索的，如果你使用之前的版本，你需要使用以下代码来启用全文检索:

```
>db.adminCommand({setParameter:true,textSearchEnabled:true})
```

或者使用命令：

```
mongod --setParameter textSearchEnabled=true
```

## 创建全文索引

考虑以下 posts 集合的文档数据，包含了文章内容（post_text）及标签(tags)：

```
{
   "post_text": "enjoy the mongodb articles on w3cschool.cc",
   "tags": [
      "mongodb",
      "w3cschool"
   ]
}
```

我们可以对 post_text 字段建立全文索引，这样我们可以搜索文章内的内容：

```
>db.posts.ensureIndex({post_text:"text"})
```

# 使用全文索引

现在我们已经对 post_text 建立了全文索引，我们可以搜索文章中的关键词w3cschool.cc：

```
>db.posts.find({$text:{$search:"w3cschool.cc"}})
```

以下命令返回了如下包含w3cschool.cc关键词的文档数据：

```
{
   "_id" : ObjectId("53493d14d852429c10000002"),
   "post_text" : "enjoy the mongodb articles on w3cschool.cc",
   "tags" : [ "mongodb", "w3cschool" ]
}
{
   "_id" : ObjectId("53493d1fd852429c10000003"),
   "post_text" : "writing tutorials on w3cschool.cc",
   "tags" : [ "mongodb", "tutorial" ]
}
```

如果你使用的是旧版本的MongoDB，你可以使用以下命令：

```
>db.posts.runCommand("text",{search:" w3cschool.cc"})
```

使用全文索引可以提高搜索效率。

# 删除全文索引

删除已存在的全文索引，可以使用 find 命令查找索引名：

```
>db.posts.getIndexes()
```

通过以上命令获取索引名，本例的索引名为post_text_text，执行以下命令来删除索引：

```
>db.posts.dropIndex("post_text_text")
```

# MongoDB 正则表达式

正则表达式是使用单个字符串来描述、匹配一系列符合某个句法规则的字符串。

许多程序设计语言都支持利用正则表达式进行字符串操作。

MongoDB 使用 **$regex** 操作符来设置匹配字符串的正则表达式。

MongoDB使用PCRE (Perl Compatible Regular Expression) 作为正则表达式语言。

不同于全文检索，我们使用正则表达式不需要做任何配置。

考虑以下 **posts** 集合的文档结构，该文档包含了文章内容和标签：

```
{
   "post_text": "enjoy the mongodb articles on tutorialspoint",
   "tags": [
      "mongodb",
      "tutorialspoint"
   ]
}
```

## 使用正则表达式

以下命令使用正则表达式查找包含 w3cschool.cc 字符串的文章：

```
>db.posts.find({post_text:{$regex:"w3cschool.cc"}})
```

以上查询也可以写为：

```
>db.posts.find({post_text:/w3cschool.cc/})
```

## 不区分大小写的正则表达式

如果检索需要不区分大小写，我们可以设置 $options 为 $i。

以下命令将查找不区分大小写的字符串 w3cschool.cc：

```
>db.posts.find({post_text:{$regex:"w3cschool.cc",$options:"$i"}})
```

集合中会返回所有包含字符串 w3cschool.cc 的数据，且不区分大小写：

```
{
   "_id" : ObjectId("53493d37d852429c10000004"),
   "post_text" : "hey! this is my post on  W3Cschool.cc",
   "tags" : [ "tutorialspoint" ]
}
```

# 数组元素使用正则表达式

我们还可以在数组字段中使用正则表达式来查找内容。 这在标签的实现上非常有用，如果你需要查找包含以 tutorial 开头的标签数据(tutorial 或 tutorials 或 tutorialpoint 或 tutorialphp)，你可以使用以下代码：

```
>db.posts.find({tags:{$regex:"tutorial"}})
```

# 优化正则表达式查询

- 如果你的文档中字段设置了索引，那么使用索引相比于正则表达式匹配查找所有的数据查询速度更快。
- 如果正则表达式是前缀表达式，所有匹配的数据将以指定的前缀字符串为开始。例如：如果正则表达式为 **^tut** ，查询语句将查找以 tut 为开头的字符串。

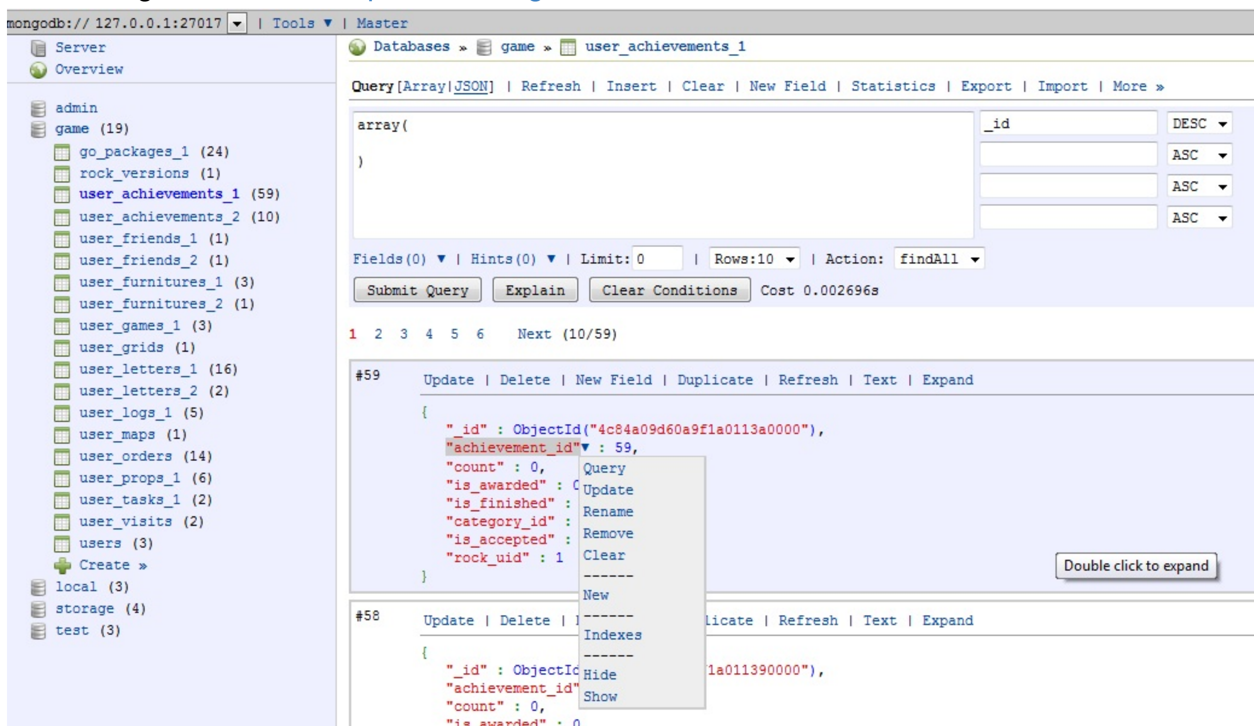
# MongoDB 管理工具: Rockmongo

RockMongo是PHP5写的一个MongoDB管理工具。

通过 Rockmongo 你可以管理 MongoDB服务，数据库，集合，文档，索引等等。

它提供了非常人性化的操作。类似 phpMyAdmin（PHP开发的MySql管理工具）。

Rockmongo 下载地址：http://rockmongo.com/downloads




## 简介

主要特征：

- 使用宽松的New BSD License协议
- 速度快，安装简单
- 支持多语言（目前提供中文、英文、日文、巴西葡萄牙语、法语、德语、俄语、意大利语）
- 系统
    - 可以配置多个主机，每个主机可以有多个管理员
    - 需要管理员密码才能登入操作，确保数据库的安全性
- 服务器
    - 服务器信息 (WEB服务器, PHP, PHP.ini相关指令 ...)
    - 状态
    - 数据库信息

- 数据库
    - 查询，创建和删除
    - 执行命令和Javascript代码
    - 统计信息
- 集合（相当于表）
    - 强大的查询工具
    - 读数据，写数据，更改数据，复制数据，删除数据
    - 查询、创建和删除索引
    - 清空数据
    - 批量删除和更改数据
    - 统计信息
- GridFS
    - 查看分块
    - 下载文件

# 安装

## 需求

- 一个能运行PHP的Web服务器，比如Apache Httpd, Nginx ...
- PHP - 需要PHP v5.1.6或更高版本，需要支持SESSION
    - 为了能连接MongoDB，你需要安装php_mongo扩展

## 快速安装

- 下载安装包
- 解压到你的网站目录下
- 用编辑器打开config.php，修改host, port, admins等参数
- 在浏览器中访问index.php，比如说：http://localhost/rockmongo/index.php
- 使用用户名和密码登录，默认为"admin"和"admin"
- 开始玩转MongoDB!

参考文章：http://rockmongo.com/wiki/introduction?lang=zh_cn

# MongoDB GridFS

GridFS 用于存储和恢复那些超过16M（BSON文件限制）的文件(如：图片、音频、视频等)。

GridFS 也是文件存储的一种方式，但是它是存储在MonoDB的集合中。

GridFS 可以更好的存储大于16M的文件。

GridFS 会将大文件对象分割成多个小的chunk(文件片段),一般为256k/个,每个chunk将作为MongoDB的一个文档(document)被存储在chunks集合中。

GridFS 用两个集合来存储一个文件：fs.files与fs.chunks。

每个文件的实际内容被存在chunks(二进制数据)中,和文件有关的meta数据(filename,content_type,还有用户自定义的属性)将会被存在files集合中。

以下是简单的 fs.files 集合文档：

```
{
   "filename": "test.txt",
   "chunkSize": NumberInt(261120),
   "uploadDate": ISODate("2014-04-13T11:32:33.557Z"),
   "md5": "7b762939321e146569b07f72c62cca4f",
   "length": NumberInt(646)
}
```

以下是简单的 fs.chunks 集合文档：

```
{
   "files_id": ObjectId("534a75d19f54bfec8a2fe44b"),
   "n": NumberInt(0),
   "data": "Mongo Binary Data"
}
```

# GridFS 添加文件

现在我们使用 GridFS 的 put 命令来存储 mp3 文件。 调用 MongoDB 安装目录下bin的mongofiles.exe工具。

打开命令提示符，进入到MongoDB的安装目录的bin目录中，找到mongofiles.exe，并输入下面的代码：

```
>mongofiles.exe -d gridfs put song.mp3
```

GridFS 是存储文件的数据名称。如果不存在该数据库，MongoDB会自动创建。Song.mp3 是音频文件名。

使用以下命令来查看数据库中文件的文档：

```
>db.fs.files.find()
```

以上命令执行后返回以下文档数据：

```
{
   _id: ObjectId('534a811bf8b4aa4d33fdf94d'),
   filename: "song.mp3",
   chunkSize: 261120,
   uploadDate: new Date(1397391643474), md5: "e4f53379c909f7bed2e9d631e15c1c41",
   length: 10401959
}
```

我们可以看到 fs.chunks 集合中所有的区块，以下我们得到了文件的 _id 值，我们可以根据这个 _id 获取区块(chunk)的数据：

```
>db.fs.chunks.find({files_id:ObjectId('534a811bf8b4aa4d33fdf94d')})
```

以上实例中，查询返回了 40 个文档的数据，意味着mp3文件被存储在40个区块中。

# MongoDB 固定集合（Capped Collections）

MongoDB 固定集合（Capped Collections）是性能出色的有着固定大小的集合，对于大小固定，我们可以想象其就像一个环形队列，当集合空间用完后，再插入的元素就会覆盖最初始的头部的元素！

## 创建固定集合

我们通过createCollection来创建一个固定集合，且capped选项设置为true：

```
>db.createCollection("cappedLogCollection",{capped:true,size:10000})
```

还可以指定文档个数,加上max:1000属性：

```
>db.createCollection("cappedLogCollection",{capped:true,size:10000,max:1000})
```

判断集合是否为固定集合:

```
>db.cappedLogCollection.isCapped()
```

如果需要将已存在的集合转换为固定集合可以使用以下命令：

```
>db.runCommand({"convertToCapped":"posts",size:10000})
```

以上代码将我们已存在的 posts 集合转换为固定集合。

## 固定集合查询

固定集合文档按照插入顺序储存的,默认情况下查询就是按照插入顺序返回的,也可以使用$natural调整返回顺序。

```
>db.cappedLogCollection.find().sort({$natural:-1})
```

## 固定集合的功能特点

可以插入及更新,但更新不能超出collection的大小,否则更新失败,不允许删除,但是可以调用drop()删除集合中的所有行,但是drop后需要显式地重建集合。

在32位机子上一个cappped collection的最大值约为482.5M,64位上只受系统文件大小的限制。

# 固定集合属性及用法

## 属性

- 属性1:对固定集合进行插入速度极快
- 属性2:按照插入顺序的查询输出速度极快
- 属性3:能够在插入最新数据时,淘汰最早的数据

## 用法

- 用法1:储存日志信息
- 用法2:缓存一些少量的文档

# MongoDB 自动增长

MongoDB 没有像 SQL 一样有自动增长的功能， MongoDB 的 _id 是系统自动生成的12字节唯一标识。

但在某些情况下，我们可以需要实现 ObjectId 实现自动增长功能。

由于 MongoDB 没有实现这个功能，我们可以通过编程的方式来实现，以下我们将在 counters 集合中实现_id字段自动增长。

## 使用 counters 集合

考虑以下 products 文档。我们希望 _id 字段实现 从 1,2,3,4 到 n 的自动增长功能。

```
{
  "_id":1,
  "product_name": "Apple iPhone",
  "category": "mobiles"
}
```

为此，创建 counters 集合，序列字段值可以实现自动长：

```
>db.createCollection("counters")
```

现在我们向 counters 集合中插入以下文档，使用 productid 作为 key:

```
{
  "_id":"productid",
  "sequence_value": 0
}
```

sequence_value 字段是序列的是通过自动增长后的一个值。

使用以下命令插入 counters 集合的序列文档中：

```
>db.counters.insert({_id:"productid",sequence_value:0})
```

## 创建 Javascript 函数

现在，我们创建函数 getNextSequenceValue 来作为序列名的输入， 指定的序列会自动增长 1 并返回最新序列值。在本文的实例中序列名为 productid 。

```
>function getNextSequenceValue(sequenceName){
   var sequenceDocument = db.counters.findAndModify(
      {
         query:{_id: sequenceName },
         update: {$inc:{sequence_value:1}},
         new:true
      });
   return sequenceDocument.sequence_value;
}
```

# 使用 Javascript 函数

接下来我们将使用 getNextSequenceValue 函数创建一个新的文档， 并设置文档 _id 自动为返回的序列值：

```
>db.products.insert({
   "_id":getNextSequenceValue("productid"),
   "product_name":"Apple iPhone",
   "category":"mobiles"})

>db.products.insert({
   "_id":getNextSequenceValue("productid"),
   "product_name":"Samsung S3",
   "category":"mobiles"})
```

就如你所看到的，我们使用 getNextSequenceValue 函数来设置 _id 字段。

为了验证函数是否有效，我们可以使用以下命令读取文档：

```
>db.prodcuts.find()
```

以上命令将返回以下结果，我们发现 _id 字段是自增长的：

```
{ "_id" : 1, "product_name" : "Apple iPhone", "category" : "mobiles"}

{ "_id" : 2, "product_name" : "Samsung S3", "category" : "mobiles" }
```

# 免责声明

W3School提供的内容仅用于培训。我们不保证内容的正确性。通过使用本站内容随之而来的风险与本站无关。W3School简体中文版的所有内容仅供测试，对任何法律问题及风险不承担任何责任。